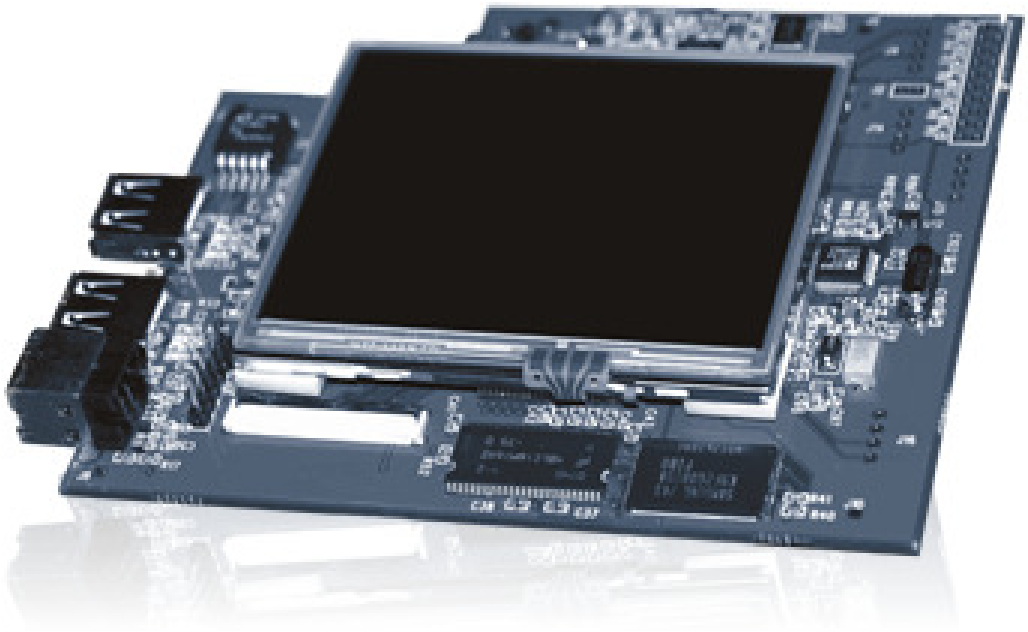
Gemstone

PE-SBC-9263-CE-1-1A

User Manual

Version 1.0



PROELEC
Professional Electronics

April 2009

Proelec has made every attempt to ensure that the information in this user manual is accurate and complete. However, Proelec assumes no responsibility for any errors, omissions, or for any consequences resulting from the use of the information included herein or the equipment it accompanies. Proelec reserves the right to make changes in its products and specifications at any time without notice.

Any software described in this document is furnished under a license or non-disclosure agreement. It is against the law to copy this software on magnetic tape, disk, or other medium for any purpose other than the licensee's personal use.

Proelec AB
Granvägen 1
82635 Söderhamn
Sweden
Tel: +46-270-76507
+46-70-5146507
Internet: www.proelec.se
E-Mail info@proelec.se

Acknowledgments:

ARM, ARM9 and Thumb are trademarks of ARM Ltd.
Windows, Win32, Windows CE are trademarks of Microsoft Corporation.
All other trademarks are trademarks of their respective companies.

Copyright © 2009 PROELEC AB
All rights reserved.

Contents

1 Document Presentation 6
1.1 Abbrevations Used 6
1.2 Purpose 6
2 General Information 7
2.1 Board Features 7
2.2 AT91SAM9263 Microcontroller Specification 7
2.3 Electrostatic Warning 9
3 Hardware 10
3.1 Layout 10
3.2 Connectors 11
3.3 Power Supply 17
3.4 Temperature Range 17
4 Windows Embedded CE 18
4.1 General Information 18
4.2 Customized Windows CE Image 18
4.2.1 Developing the Application 18
4.2.2 Kiosk Mode 18
4.3 USB Driver 19
4.3.1 USB Host 19
4.3.2 USB Device 19
4.3.3 Interface 19
4.4 Display Driver 20
4.4.1 Configure the Display 20
4.4.2 Implementation 20
4.5 ADS7846 Touch screen driver 21
4.5.1 Implementation 21
4.5.2 Calibration 21
4.6 NANDFlash Driver 21
4.6.1 Integrated Support 21
4.7 10/100 Fast Ethernet Driver 22
4.8 Serial Driver 22

4.8.1 Implementation 23
4.8.2 Managed Code 23
4.9 PIO Driver 24
4.9.1 Implementation 24
4.9.2 Managed Driver 24
4.9.2.1 OutputPort 24
4.9.2.2 InputPort 25
4.9.2.3 InputEventPort 26
4.9.2.4 WakeupPort 27
4.9.2.5 PeripheralPort 28
4.9.2.6 PWMPort 29
4.10 CAN Driver 30
4.10.1 Implementation 31
4.10.2 Managed Driver 31
4.10.2.1 CANMessage 31
4.10.2.2 Config 32
4.10.2.3 TransmitMailbox 33
4.10.2.4 SimpleReceptionMailbox 34
4.10.2.5 ChainedReceptionMailbox 36
4.11 TWI/I2C Driver 38
4.11.1 Implementation 38
4.11.2 Managed Driver 39
4.11.2.1 I2CBus 39
4.12 SPI Driver 41
4.12.1 Implementation 42
4.12.2 Managed Driver 42
5 Linux 43
5.1 General Information 43
6 FlashTool 44
6.1 General Information 44
6.2 Windows CE 44
6.2.1 Program GUI 44
6.2.2 Install SAM-BA Windows Driver 45
6.2.3 Starting up the board in SAM-BA Mode 45

6.2.4 Programming the DataFlash 46
6.2.5 Erasing the DataFlash 46
6.2.6 Programming the NandFlash 46
6.2.7 Erasing the NandFlash 47
6.2.8 EBoot Settings 47
6.2.9 Programming Status 47
6.3 Linux 47
7 Appendix A: Schematics 48
8 Revision History 58
9 References 59

1 Document Presentation

1.1 Abbrevations Used

BSP	Board Support Package
OAL	OEM Adaptation Layer
PB	Platform Builder
VS	Visual Studio
WinCE	Windows CE
MDD	Model Device Driver – platform independent layer of a device driver
PDD	Platform Dependent Driver - platform dependent layer of a device driver
CF	Compact .NET Framework
HW	Hardware
SW	Software
FIFO	First In First Out Queue
FAL	Flash Abstraction Layer
FMD	Flash Media Device
MBR	Master Boot Record
GPIO	General Purpose Input/Output
PIO	Parallel Input/Output
CAN	Controller Area Network
TWI	Two-Wire Interface
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
GUI	Graphical User Interface
HIVE	A logical group of keys, subkeys, and values in the registry that has a set of supporting files containing backups of its data

1.2 Purpose

This document aims to give a description of the GEMSTONE hardware and the Windows CE 6.0 drivers, assemblies made by Proelec that can be used in VS (Visual Studio).
This document will not list any source code explaining how all the WinCE drivers are implemented in the BSP (Board Support Package). More detailed information is only available on managed drivers that can be used in managed code.

2 General Information

2.1 Board Features

The board is equipped with an AT91SAM9263 Microcontroller from Atmel (324-ball LFBGA package) together with the following:

- 64 Mbytes of SDRAM memory
- 256 Mbytes of NANDFlash memory
- One TWI bus (I2C)
- One USB USB Full Speed (12 Mbits/s) device port interface
- Two USB Full Speed (12 Mbits/s) host port interfaces
- Three RS232 serial communication ports
- One DBGU serial communication port
- One serial CAN 2.0B communication port
- One Ethernet 100-base TX with two status LEDs
- One AC97 Audio Controller
- One 3.5"-10" QVGA TFT LCD with TouchScreen and Backlight control
- One Lithium Coin Cell Battery Retainer for 12 mm cell size

2.2 AT91SAM9263 Microcontroller Specification

- Incorporates the ARM926EJ-S™ ARM® Thumb® Processor
  – DSP instruction Extensions, Jazelle® Technology for Java® Acceleration
  – 16 Kbyte Data Cache, 16 Kbyte Instruction Cache, Write Buffer
  – 200 MIPS at 180 MHz, 220 MIPS at 200 MHz
  – Memory Management Unit
  – EmbeddedICE™, Debug Communication Channel Support
  – Mid-level Implementation Embedded Trace Macrocell™
- Bus Matrix
  – Nine 32-bit-layer Matrix, Allowing a Total of 28.8 Gbps of On-chip Bus Bandwidth
  – Boot Mode Select Option, Remap Command
- Embedded Memories
  – One 128 Kbyte Internal ROM, Single-cycle Access at Maximum Bus Matrix Speed
  – One 80 Kbyte Internal SRAM, Single-cycle Access at Maximum Processor Bus Matrix Speed
  – One 16 Kbyte Internal SRAM, Single-cycle Access at Maximum Bus Matrix Speed
- Dual External Bus Interface (EBI0 and EBI1)
  – EBI0 Supports SDRAM, Static Memory, ECC-enabled NAND Flash and CompactFlash®
  – EBI1 Supports SDRAM, Static Memory and ECC-enabled NAND Flash
- DMA Controller (DMAC)
  – Acts as one Bus Matrix Master
  – Embeds 2 Unidirectional Channels with Programmable Priority, Address Generation, Channel Buffering and Control
- Twenty Peripheral DMA Controller Channels (PDC)

- LCD Controller
  – Supports Passive or Active Displays
  – Up to 24 bits per Pixel in TFT Mode, Up to 16 bits per Pixel in STN Color Mode
  – Up to 16M Colors in TFT Mode, Resolution Up to 2048x2048, Supports Virtual Screen Buffers
- 2D Graphics Accelerator
  – Line Draw, Block Transfer, Polygon Fill, Clipping, Commands Queuing
- Image Sensor Interface
  – ITU-R BT. 601/656 External Interface, Programmable Frame Capture Rate
  – 12-bit Data Interface for Support of High Sensibility Sensors
  – SAV and EAV Synchronization, Preview Path with Scaler, YCbCr Format
- USB 2.0 Full Speed (12 Mbits per second) Host Double Port
  – Dual On-chip Transceivers
  – Integrated FIFOs and Dedicated DMA Channels
  • USB 2.0 Full Speed (12 Mbits per second) Device Port
  – On-chip Transceiver, 2,432-byte Configurable Integrated DPRAM
- Ethernet MAC 10/100 Base-T
  – Media Independent Interface or Reduced Media Independent Interface
  – 28-byte FIFOs and Dedicated DMA Channels for Receive and Transmit
- Fully-featured System Controller, including
  – Reset Controller, Shutdown Controller
  – Twenty 32-bit Battery Backup Registers for a Total of 80 Bytes
  – Clock Generator and Power Management Controller
  – Advanced Interrupt Controller and Debug Unit
  – Periodic Interval Timer, Watchdog Timer and Double Real-time Timer
- Reset Controller (RSTC)
  – Based on Two Power-on Reset Cells, Reset Source Identification and Reset Output Control
- Shutdown Controller (SHDWC)
  – Programmable Shutdown Pin Control and Wake-up Circuitry
- Clock Generator (CKGR)
  – 32768Hz Low-power Oscillator on Battery Backup Power Supply, Providing a Permanent Slow Clock
  – 3 to 20 MHz On-chip Oscillator and Two Up to 240 MHz PLLs
- Power Management Controller (PMC)
  – Very Slow Clock Operating Mode, Software Programmable Power Optimization Capabilities
  – Four Programmable External Clock Signals
- Advanced Interrupt Controller (AIC)
  – Individually Maskable, Eight-level Priority, Vectored Interrupt Sources
  – Two External Interrupt Sources and One Fast Interrupt Source, Spurious Interrupt Protected
- Debug Unit (DBGU)
  – 2-wire UART and Support for Debug Communication Channel, Programmable ICE Access Prevention
- Periodic Interval Timer (PIT)
  – 20-bit Interval Timer plus 12-bit Interval Counter
- Watchdog Timer (WDT)
  – Key-protected, Programmable Only Once, Windowed 16-bit Counter Running at Slow Clock
- Two Real-time Timers (RTT)
  – 32-bit Free-running Backup Counter Running at Slow Clock with 16-bit Prescaler
- Five 32-bit Parallel Input/Output Controllers (PIOA, PIOB, PIOC, PIOD and PIOE)
  – 160 Programmable I/O Lines Multiplexed with Up to Two Peripheral I/Os

  - Input Change Interrupt Capability on Each I/O Line
  - Individually Programmable Open-drain, Pull-up Resistor and Synchronous Output
- One Part 2.0A and Part 2.0B-compliant CAN Controller
  - 16 Fully-programmable Message Object Mailboxes, 16-bit Time Stamp Counter
- Two Multimedia Card Interface (MCI)
  - SDCard/SDIO and MultiMediaCard™ Compliant
  - Automatic Protocol Control and Fast Automatic Data Transfers with PDC
  - Two SDCard Slots Support on eAch Controller
- Two Synchronous Serial Controllers (SSC)
  - Independent Clock and Frame Sync Signals for Each Receiver and Transmitter
  - I²S Analog Interface Support, Time Division Multiplex Support
  - High-speed Continuous Data Stream Capabilities with 32-bit Data Transfer
- One AC97 Controller (AC97C)
  - 6-channel Single AC97 Analog Front End Interface, Slot Assigner
- Three Universal Synchronous/Asynchronous Receiver Transmitters (USART)
  - Individual Baud Rate Generator, IrDA® Infrared Modulation/Demodulation, Manchester Encoding/Decoding
  - Support for ISO7816 T0/T1 Smart Card, Hardware Handshaking, RS485 Support
- Two Master/Slave Serial Peripheral Interface (SPI)
  - 8- to 16-bit Programmable Data Length, Four External Peripheral Chip Selects
  - Synchronous Communications at Up to 90Mbits/sec
- One Three-channel 16-bit Timer/Counters (TC)
  - Three External Clock Inputs, Two Multi-purpose I/O Pins per Channel
  - Double PWM Generation, Capture/Waveform Mode, Up/Down Capability
- One Four-channel 16-bit PWM Controller (PWMC)
- One Two-wire Interface (TWI)
  - Master Mode Support, All Two-wire Atmel® EEPROMs Supported
  - IEEE® 1149.1 JTAG Boundary Scan on All Digital Pins
- Required Power Supplies
  - 1.08V to 1.32V for VDDCORE and VDDBU
  - 3.0V to 3.6V for VDDOSC, VDDPLL and VDDIOP0 (Peripheral I/Os)
  - 1.65V to 3.6V for VDDIOP1 (Peripheral I/Os)
  - Programmable 1.65V to 1.95V or 3.0V to 3.6V for VDDIOM0/VDDIOM1 (Memory I/Os)
- Available in a 324-ball BGA Green Package

2.3 Electrostatic Warning

The board is shipped in protective anti-static packaging. The board must not be subjected to high electrostatic potentials. A grounding strap or similar protective device should be worn when handling the board. Avoid touching the component pins or any other metallic element.

3 Hardware

3.1 Layout

The board layout has 6 signal layers. Two of them are the power- and GND-plane.

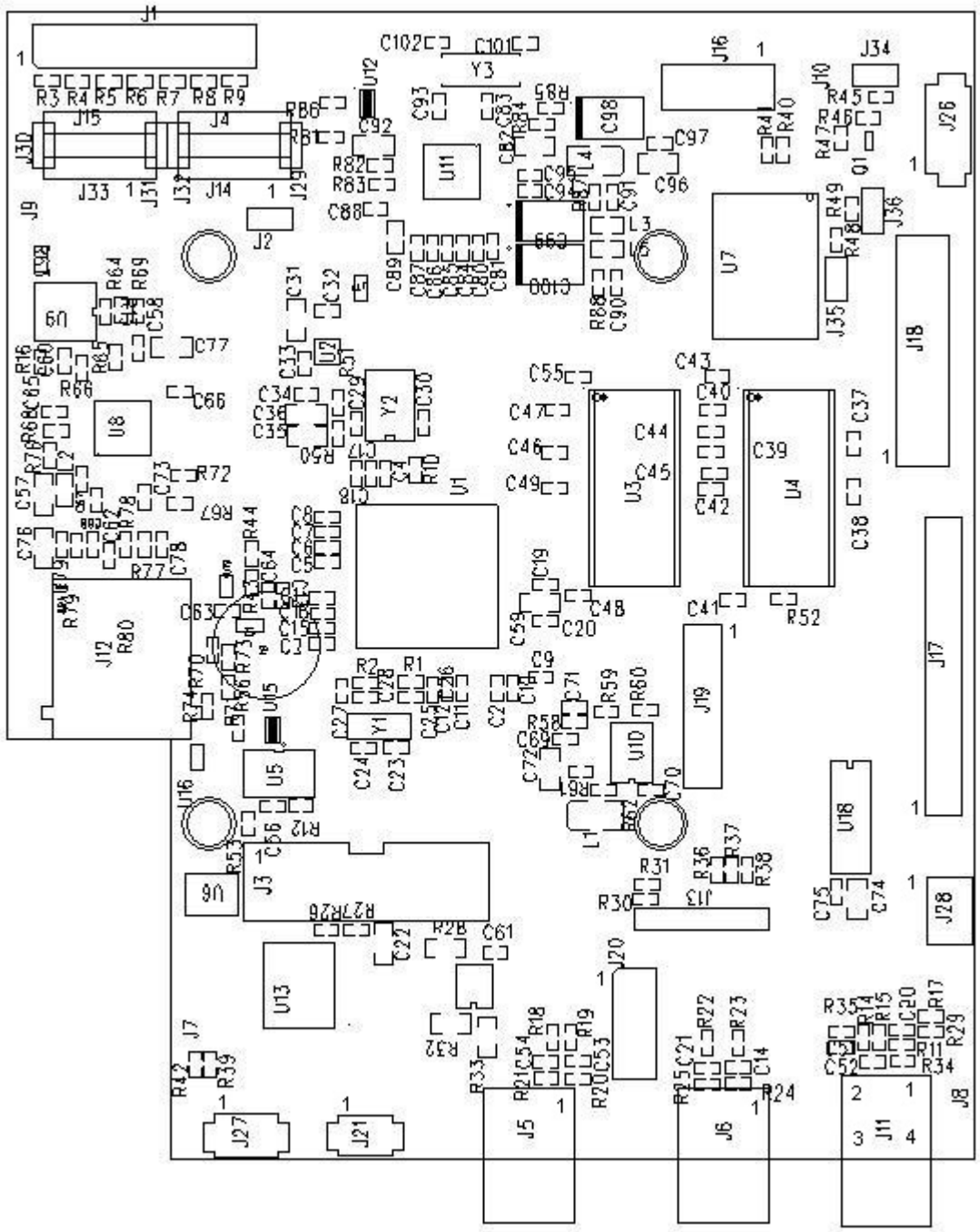


Figure 2.1

3.2 Connectors

All board connectors are shown in figure 2.1

ICE Interface	
J1-1	3V3
J1-2	3V3
J1-3	NTRST
J1-4	GND
J1-5	TDI
J1-6	GND
J1-7	TMS
J1-8	GND
J1-9	TCK
J1-10	GND
J1-11	RTCK
J1-12	GND
J1-13	TDO
J1-14	GND
J1-15	NRST
J1-16	GND
J1-17	NC
J1-18	GND
J1-19	NC
J1-20	GND

Power, GPIO and CAN	
J3-1	PB20
J3-2	PB21
J3-3	PB22
J3-4	PB23
J3-5	PB25
J3-6	PB26
J3-7	PB30
J3-8	PB31
J3-9	CAN-H
J3-10	CAN-L
J3-11	PB7
J3-12	PB29
J3-13	GND
J3-14	GND
J3-15	5V
J3-16	5V

USART 1	
J15-1	TXD, Peripheral A (PA26)
J15-2	RXD, Peripheral A (PA27)
J15-3	RTS, Peripheral A (PA28)
J15-4	CTS, Peripheral A (PA29)

USART 2	
J4-1	TXD, Peripheral A (PD0)
J4-2	RXD, Peripheral A (PD1)
J4-3	RTS, Peripheral B (PD7)
J4-4	CTS, Peripheral B (PD8)

USART 3	
J14-1	TXD, Peripheral A (PD2)
J14-2	RXD, Peripheral A (PD3)
J14-3	RTS, Peripheral B (PD5)
J14-4	CTS, Peripheral B (PD6)

USB Host A	
J5-1	GND
J5-2	HDPA
J5-3	HDMA
J5-4	5V

USB Host B	
J6-1	GND
J6-2	HDPB
J6-3	HDMB
J6-4	5V

USB Device	
J11-1	CNX
J11-2	DDM
J11-3	DDP
J11-4	GND

RJ45 ETHERNET CONNECTOR	
J12	Ethernet 100-base TX with two status LEDs

Debug UART (DBGU)	
J13-1	NC
J13-2	DTXD
J13-3	DRXD
J13-4	NC
J13-5	NC
J13-6	GND

Sound AC97	
J16-1	Headphone Left
J16-2	Headphone Right
J16-3	5V
J16-4	GND

LCD – LVDS	
J17-1	GND
J17-2	TXOUT0-
J17-3	TXOUT0+
J17-4	TXOUT1-
J17-5	TXOUT1+
J17-6	TXOUT2-
J17-7	TXOUT2+
J17-8	GND
J17-9	TXCLK0-
J17-10	TXCLK0+
J17-11	NC
J17-12	NC
J17-13	NC
J17-14	NC
J17-15	GND
J17-16	NC
J17-17	NC
J17-18	GND
J17-19	NC
J17-20	NC
J17-21	NC
J17-22	NC
J17-23	NC
J17-24	NC
J17-25	GND
J17-26	NC
J17-27	NC
J17-28	NC
J17-29	3V3

J17-30	3V3
J17-31	3V3
J17-32	NC

LCD	
J18-1	5V
J18-2	5V
J18-3	GND
J18-4	GND
J18-5	3V3
J18-6	3V3
J18-7	Backlight ON/OFF (PB9)
J18-8	GND
J18-9	NC
J18-10	NC
J18-11	B0 (PC22)
J18-12	B1 (PC23)
J18-13	B2 (PC24)
J18-14	B3 (PC17)
J18-15	B4 (PC26)
J18-16	B5 (PC27)
J18-17	NC
J18-18	NC
J18-19	G0 (PC14)
J18-20	G1 (PC15)
J18-21	G2 (PC16)
J18-22	G3 (PC12)
J18-23	G4 (PC18)
J18-24	G5 (PC19)
J18-25	NC
J18-26	NC
J18-27	R0 (PC6)
J18-28	R1 (PC7)
J18-29	R2 (PC8)
J18-30	R3 (PC9)
J18-31	R4 (PC10)
J18-32	R5 (PC11)
J18-33	GND
J18-34	GND
J18-35	DOTCK (PC2)
J18-36	VSYNC (PC0)
J18-37	DATA ENABLE (PC3)
J18-38	HSYNC (PC1)
J18-39	3V3
J18-40	3V3

LCD	
J19-1	3V3
J19-2	3V3
J19-3	3V3
J19-4	DOTCK (PC2)
J19-5	GND
J19-6	HSYNC (PC1)
J19-7	GND
J19-8	DATA ENABLE (PC3)
J19-9	GND
J19-10	NC
J19-11	GND
J19-12	R5 (PC11)
J19-13	R4 (PC10)
J19-14	R3 (PC9)
J19-15	GND
J19-16	R2 (PC8)
J19-17	R1 (PC7)
J19-18	R0 (PC6)
J19-19	NC
J19-20	G5 (PC19)
J19-21	G4 (PC18)
J19-22	G3 (PC12)
J19-23	NC
J19-24	G2 (PC16)
J19-25	G1 (PC15)
J19-26	G0 (PC14)
J19-27	NC
J19-28	B5 (PC27)
J19-29	B4 (PC26)
J19-30	B3 (PC17)
J19-31	NC
J19-32	B2 (PC24)
J19-33	B1 (PC23)
J19-34	B0 (PC22)
J19-35	Power Control In (3V3)
J19-36	Contrast Control (3V3 or PB9)
J19-37	X-RIGHT (Touchscreen)
J19-38	Y-LOW (Touchscreen)
J19-39	X-LEFT (Touchscreen)
J19-40	Y-UP (Touchscreen)

USB	
J20-1	5V
J20-2	GND
J20-3	HDMA (Host A)
J20-4	GND
J20-5	HDPA (Host A)
J20-6	HDPB (Host B)
J20-7	GND
J20-8	HDMB (Host B)
J20-9	GND
J20-10	5V

Power Supply	
J21-1	5V
J21-2	12V (Inverter J26)
J21-3	GND
J21-4	GND

LCD Inverter	
J26-1	Contrast Control
J26-2	Contrast Control (PB8)
J26-3	Backlight ON/OFF (PB9)
J26-4	GND
J26-5	GND
J26-6	12V
J26-7	12V

TWI (I2C)	
J27-1	TWD (SDA)
J27-2	TWCK (SCL)
J27-3	GND
J27-4	GND
J27-5	5V

Touchscreen	
J28-1	X-LEFT
J28-2	Y-LOW
J28-3	X-RIGHT
J28-4	Y-UP

3.3 Power Supply

Connect 5V and ground (GND) from the power supply to J21.
The 12V power supply on J21 is only used if a display inverter is connected to J26.

3.4 Temperature Range

The GEMSTONE platform is available in two temperature ranges:

- Standard temperature range -20° +70°C
- Extreme industrial temperature range -40° +85°C

Contact Proelec for more information about the industrial variant.

4 Windows Embedded CE

4.1 General Information

The GEMSTONE platform and Windows Embedded CE 6.0 can be used in many types of devices. One typical device can be a real-time and mission-critical industrial controller with display support. The platform is very usable when you want to cooperate with a system in a machine and a display is needed to interact with the user. The display is also usable in systems there you want to show different type of sensor data. The touch screen support also makes it very easy to interact with the system without a keypad.

4.2 Customized Windows CE Image

The Windows Embedded CE 6.0 image (kernel + BSP) is built with PB (Platform Builder), an integrated tool in VS 2005. The image can be customized to support all type of functions and peripherals on the board. Please contact Proelec for more information about building a customized WinCE image.

4.2.1 Developing the Application

When you create a new application (VS project) it's possible to use a custom image from Proelec that has Windows Explorer and ActiveSync enabled. These features make it possible to deploy and debug the application on the device directly from VS.

Requirements:

- GEMSTONE hardware, PE-SBC-9263-CE-1-1A
- Windows Embedded CE 6.0 image from Proelec (developer version)
- A PC with Windows XP or Windows Vista
- Microsoft Visual Studio 2005 + available service packs from Microsoft
- Compact .NET Framework 2.0 + available service packs from Microsoft
- ActiveSync enabled on the developer PC.
- Platform SDK from Proelec
- FlashTool from Proelec or SAM-BA from Atmel (used when programming the DataFlash/NandFlash on the board)
- USB Device cable

4.2.2 Kiosk Mode

When the application is finished and is going to be released to customer, a new WinCE image can be made by Proelec that disables Windows Explorer and instead starts the application (kiosk mode). The ActiveSync feature can also be disabled for higher security. Customized start-up code in C/C++ can also be written to support application updates via USB or other needed start-up functionality.

4.3 USB Driver

4.3.1 USB Host

USB host controllers mostly conform to one of the two standard specifications:

- OHCI
- UHCI

WinCE provides support for both this kind of controllers. All the processing is done by the Model Device Driver (MDD) that exposes a stream driver interface. The Platform Dependent Driver (PDD) only turns on the OHCI controller and initializes some variables (address of the controller, system interrupt number and some objects specific to the OHCI MDD).
The driver works with USB keyboards, mouse and Mass Storage Devices.

4.3.2 USB Device

This driver is also known as an USB Function driver. Its primary purpose is to provide connectivity between a WinCE device and a desktop PC.
The idea is that a WinCE device that contains suitable USB Function controller hardware will be used as a serial port, mass storage device or a network device (RNDIS) when it's connected to a desktop PC. Essentially, there are two drivers that will be used in this type of designs:

- **USB Function driver**
  Runs on the WinCE platform and exposes the device as the wanted device to the desktop
- **Host-side USB driver**
  Runs on the desktop PC and talks to the WinCE via USB transfers.
  The host driver also exposes a serial interface to the application layer so that a client application (such as ActiveSync) running on the desktop can use it for any serial applications.

4.3.3 Interface

Communication over the USB is done by sending packets of data on the same physical line. However USB (as a protocol) provides multiple logical lines. Those lines are called endpoints.
For more information about USB, please have a look at
http://www.beyondlogic.org/usbnutshell/usb1.htm.

4.4 Display Driver

The WinCE display driver is built on a layered architecture. The display driver is loaded by the *Graphics, Windowing and Events Subsystem* (*GWES*) at WinCE boot. *GWES* calls *DDI.dll* that is the default dll for the display driver. This dll exports a single function called *DrvEnableDriver* that returns to the caller a pointer to an array of 27 function pointers. Those functions are called as soon as GWES needs to display something on the device.

4.4.1 Configure the Display

When a display is configured to be used in our platform the parameters for the display and screen are stored in the Windows registry. This feature makes it very easy to handle many types of different displays that can be used with our platform.
Please contact Proelec for more information regarding different types of display solutions.

Example of settings in the file lcd.reg to handle a Toshiba 800x600 LVDS display:

```
[HKEY_LOCAL_MACHINE\System\GDI\ROTATION]
     "Angle"=dword:0

[HKEY_LOCAL_MACHINE\Drivers\Display\LCDC]
     "Width"=dword:320                    ;Screen width in pixel
     "Height"=dword:258                   ;Screen heigth in pixel
     "Bpp"=dword:10                       ;Bit per pixel
     "VRAMWidthInPixel"=dword:400         ;Frame buffer width in pixel
     "VRAMHeightInPixel"=dword:400        ;Frame buffer heigth in
     "VRAMaddress"=dword:23e00000         ;Frame buffer location.
     "VRAMBusWidth"=dword:20              ;SDRAM has a 32-bits bus width
     "UpperMargin"=dword:10               ; Vertical Back Porch
     "LowerMargin"=dword:10               ; Vertical Front Porch
     "LeftMargin"=dword:10                ; Horizontal Back Porch
     "RightMargin"=dword:10               ; Horizontal Front Porch
     "Vsync"=dword:2                      ; Vertical Sync pulse width
     "Hsync"=dword:8                      ; Horizontal sync pulse width
     "PixelClock"=dword:1312D00           ; Pixel clock
```

4.4.2 Implementation

It uses Graphics Primitive Engine (GPE) to provide a fast and reliable interface with Hardware (HW), implements DirectDraw and provide a D3DMobile support. The driver is designed to use a rectangle video memory space, placed either in general RAM or in specific Video RAM. The driver permits to select which operation you want to do with HW, and which you want to do in Software (SW).
All HW calls are asynchronous and push to a FIFO. The driver support all display mode supported by the GPE.

4.5 ADS7846 Touch screen driver

The touchpad driver is a layered native driver. Layered means that it's composed of two layers.
The upper one is the platform independent MDD layer which implements the interface with the system and is the functional part of the driver.
The second one is the platform dependant PDD layer that implements the interface to the hardware.
Native means that this driver has a specific interface dedicated to touch screen drivers.

4.5.1 Implementation

The ADS7846 driver uses the SPI driver for its communication, and use specific IOControls to perform transaction between the chip and the driver. A timer forces input detection to perform the sampling for the detection of stylus movement.

4.5.2 Calibration

At startup, a calibration application is launched allowing users to calibrate the touch screen if no corresponding information is found in the registry. If Hive based registry is used for the device, the calibration is only performed once at the first device boot.

Registry entries needed to load the driver at startup:

```
[HKEY_LOCAL_MACHINE\init]
    "Launch70"="touchcalibrate1.exe"
    "Depend70"=hex:14,00, 1e,00
```

4.6 NANDFlash Driver

The NAND flash driver is a block driver separated in two layers. The Flash Abstraction Layer (FAL) is oriented toward the system and exposes the block driver interface. It relies on the Flash Media Device (FMD) to access the hardware. The FAL also manages the wear leveling.
This layer is provided by Microsoft.
FMD layer handles the hardware transactions. It is provided by the OEM. It basically consists in a set of functions to read, erase and write sectors. On our platform, the FMD has been enhanced to add another level of translation and protection. This allows us among other things to set the Master Boot Record (MBR) at any physical address instead of sector #0. This FMD is called the Generic FMD.

4.6.1 Integrated Support

The board has 256MB NAND Flash that can be used to store the WinCE image and to offer a logical device that can be used to store files.
If Hive based registry is enabled, the device is storing the registry settings.

4.7 10/100 Fast Ethernet Driver

The driver supports the EmacB controller on the AT91SAM9263. This internal controller takes place in the Ethernet stack as a MAC controller. The driver implements the Network Device Interface (NDIS). NDIS is part of the networking architecture used in Microsoft Windows operating systems. NDIS provides simplified miniport device driver architecture to enable communication with network adapters using common driver interfaces.

4.8 Serial Driver

This is a layered driver. The driver is implemented as a stream driver. Stream means that this driver is accessed through a Win32 file IO interface.

The standard stream driver functions exported by the driver are:
COM_Init
COM_Deinit
COM_Open
COM_Close
COM_Read
COM_Write
COM_Seek
COM_PowerDown
COM_PowerUp
COM_IOControl

The received data is stocked in a specific class named “DmaManager”. The DMAManager is composed of two FIFO and a current buffer, see figure 4.1.

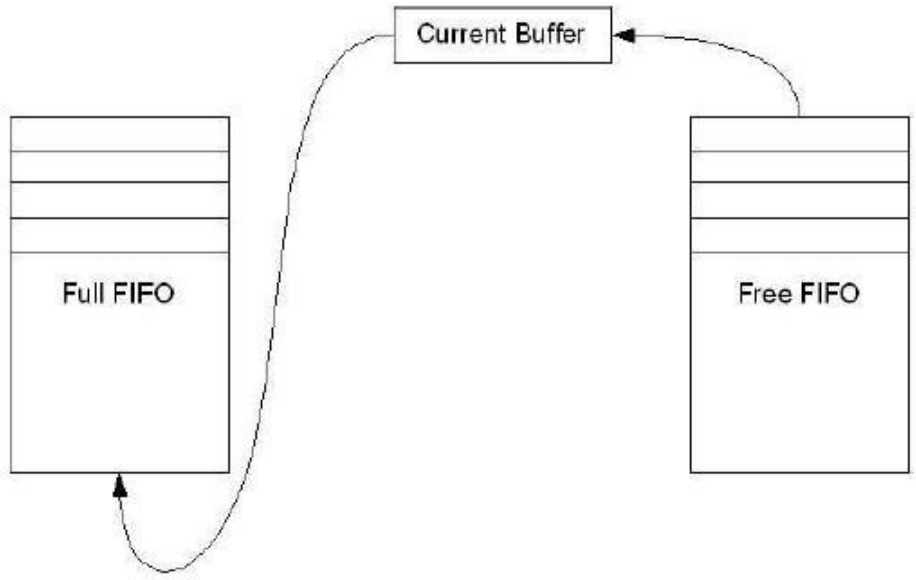


Figure 4.1

4.8.1 Implementation

Reception buffers' number and size and emission buffers' size are given by the registry. There are no software imposed limits for the size. There must be at least 4 buffers for reception. Currently the driver uses 2 buffers for emission.
During emission, the driver uses the 2 buffers in a round robin fashion always feeding the PDC with a filled buffer and pre filling the other one until there is no more data to send. Feeding the PDC means updating its registers with the address and filled size of a DMA buffer containing bytes to send.
Once the PDC transmitted those bytes to the UART, it launches an empty buffer interrupt handled by the driver by feeding the PDC with its ready buffer and by refilling the buffer just released.
During reception, the driver uses its available buffers through the DMA manager as explained at the beginning of this chapter. When flow control is activated, the RTS line is cleared when there are more than a half of the available buffers free and raised when there is more than three quarters of the available buffers full.

4.8.2 Managed Code

In managed code you can use the SerialPort class that exists in Compact .NET Framework (CF). The class supports the serial driver in WinCE. A custom driver from Proelec can also be used if you have any problem with the SerialPort class. Contact Proelec for more information.

4.9 PIO Driver

The AT91SAM9263 Parallell Input/Output (PIO) controller can manage up to 32 fully programmable input/output (I/O) lines. Each I/O line may be dedicated as a general-purpose I/O (GPIO) or be assigned to a special function of an embedded peripheral. This assures effective optimization of the pins.

Each I/O line of the PIO Controller features:
• An input change interrupt enabling level change detection on any I/O line.
• A glitch filter providing rejection of pulses lower than one-half of clock cycle.
• Multi-drive capability similar to an open drain I/O line.
• Control of the the pull-up of the I/O line.
• Input visibility and output control.
The PIO Controller also features a synchronous output providing up to 32 bits of data output in a single write operation.

4.9.1 Implementation

The driver is implemented as a stream driver, and the driver is accessed through a Win32 file IO interface in WinCE. A managed driver is available that can be used in CF and VS to handle different types of functions on the available PIO pins that exists on the board.

4.9.2 Managed Driver

The driver is an assembly developed by Proelec that can be used in both C# and VB.NET.
The assembly help file is available in the distribution package, but can also be downloaded from the Proelec homepage.

Filename: PEWinCEGPIOLib.dll
Namespace: Proelec.Win32CE.IO.GPIO
Help file: PEWinCEGPIOLib.chm

4.9.2.1 OutputPort

A class that represents an output pin on one of the PIO controller ports.

***Constructor*:**

```
[C#]
public void OutputPort(
   Port port,
   UInt32 pinNumber,
   bool initialState
);
```

Parameters:

port : Port controller, Enum: {PA, PB, PC, PD, PE}
pinNumber: Pin number to use on the port controller
initialState: Initial state of the pin, true (high) or false (low)

Example:
Sets pin PC9 as an output and the initial state to true.

```
OutputPort port = new OutputPort(Port.PC, 9, true);
```

Properties:

State: Returns or sets the output state

```
public bool State {get; set;}
```

Methods:

Set: Sets the output state to true. Returns true if success, else false

```
public bool Set();
```

Clear: Sets the output state to false. Returns true if success, else false

```
public bool Clear();
```

Read: Reads and returns the output state

```
public bool Read();
```

Write: Sets the output state. Returns true if success, else false

```
public bool Write(bool state);
```

Toogle: Toggles the output state. Returns true if success, else false

```
public bool Toogle();
```

4.9.2.2 InputPort

A class that represents an input pin on one of the PIO controller ports.

Constructor:

```
[C#]
public void InputPort(
   Port port,
   UInt32 pinNumber,
);

public void InputPort(
   Port port,
   UInt32 pinNumber,
   bool pullUp,
   bool glitchFilter
);
```

Parameters:

port : Port controller, Enum: {PA, PB, PC, PD, PE}

pinNumber: Pin number to use on the port controller

pullUp: Enables or disables the internal pull-up resistor, default = true

glitchFilter: Enables or disables the internal glitch filter, default = false

Example:

Sets pin PB20 as an input.

```
InputPort port = new InputPort(Port.PB, 20);
```

Properties:
State: Returns the input pin state
```
public bool State {get;}
```

Pullup: Returns or Sets the internal pull-up resistor enable property
```
public bool Pullup {get; set;}
```

GlitchFilter: Returns or Sets the internal glitch filter enable property
```
public bool GlitchFilter {get; set;}
```

Methods:
Read: Reads and returns the input state
```
public bool Read();
```

4.9.2.3 InputEventPort

A class that represents an event driven input pin on one of the PIO controller ports.

Constructor:
```
[C#]
public void InputEventPort(
   Port port,
   UInt32 pinNumber,
);

public void InputEventPort(
   Port port,
   UInt32 pinNumber,
   bool pullUp,
   bool glitchFilter
   CEThreadPriority priority,
   bool enabled
);
```

Parameters:

port :	Port controller, Enum: {PA, PB, PC, PD, PE}
pinNumber:	Pin number to use on the port controller
pullUp:	Enables or disables the internal pull-up resistor, default = true
glitchFilter:	Enables or disables the internal glitch filter, default = false
priority:	Windows CE thread priority, default = CEThreadPriority.Normal
enabled:	Enables or disables event handler on changed input state, default = true

Example:
Sets pin PB20 as an event driven input.
```
InputEventPort port = new InputEventPort(Port.PB, 20);
```

Properties:

State: Returns the input pin state
```
public bool State {get;}
```

Enabled: Enables or disables event handler on changed input state
```
public bool Enabled {get; set;}
```

Methods:

Read: Reads and returns the input state
```
public bool Read();
```

WndProc: The window process message method
```
protected override void WndProc(
   ref Message msg
);
```

Dispose: The dispose method used to free allocated resources in WinCE
```
new public void Dispose();
```

Events:

StateChanged: Event is trigged when the state is changed on the selected pin
```
public event System.EventHandler StateChanged;
```

StateEventArguments: The state event argument constructor
```
public void StateEventArgs(
   bool state
);
```

PortState property: Returns the current pin state
```
public bool PortState {get;}
```

4.9.2.4 WakeupPort

A class that represents a wake-up pin on one of the PIO controller ports.

The selected pin can be used to wake-up the system after suspending WinCE.

Constructor:
```
[C#]
public void WakeupPort(
   Port port,
   UInt32 pinNumber,
);

public void WakeupPort(
   Port port,
   UInt32 pinNumber,
   bool pullUp,
   bool glitchFilter
);
```

Parameters:

port : Port controller, Enum: {PA, PB, PC, PD, PE}

pinNumber: Pin number to use on the port controller

pullUp: Enables or disables the internal pull-up resistor, default = true
glitchFilter: Enables or disables the internal glitch filter, default = false

Example:
Register PC4 as an wake-up pin.

```
WakeupPort port = new WakeupPort(Port.PC, 4);
```

Properties:

State: Returns the wake-up pin state

```
public bool State {get;}
```

Pullup: Returns or Sets the internal pull-up resistor enable property

```
public bool Pullup {get; set;}
```

GlitchFilter: Returns or Sets the internal glitch filter enable property

```
public bool GlitchFilter {get; set;}
```

Methods:

Read: Reads and returns the wake-up state

```
public bool Read();
```

Remove: Unregisters the pin as an wake-up

```
public bool Remove();
```

4.9.2.5 PeripheralPort

A class that is used to set a pin on one of the PIO controller ports to one of the peripheral functions, A or B.

Constructor:

```
[C#]
public void PeripheralPort(
   Port port,
   UInt32 pinNumber,
   PeripheralGroup peripheral
);

public void PeripheralPort(
   Port port,
   UInt32 pinNumber,
   PeripheralGroup peripheral,
   bool pullUp,
   bool openDrain
);
```

Parameters:

port : Port controller, Enum: {PA, PB, PC, PD, PE}
pinNumber: Pin number to use on the port controller
peripheral : Peripheral function, Enum: {A, B}

pullUp: Enables or disables the internal pull-up resistor, default = false
openDrain: Enables or disables the internal open drain transistor, default = true

Example:
Set pin PD0 to peripheral function TXD1 (USART 1).

```
PeripheralPort port = new PeripheralPort(Port.PD, 0, PeripheralGroup.A);
```

Properties:

State: Returns the pin state

```
public bool State {get;}
```

Methods:

Read: Reads and returns the wake-up state

```
public bool Read();
```

4.9.2.6 PWMPort

A class that represents a PWM-channel.

Constructor:

```
[C#]
public void PWMPort(
   PWMChannel channel,
   UInt32 frequency,
   UInt32 dutyCycle
);
```

Parameters:

channel : PWM channel, Enum: {PWC0, PWC1, PWC2, PWC3}
frequency: The frequency in Hz
dutyCycle : The duty cycle in % (0-100)

Example:
Using PC28 as PWM channel 0, frequency 1KHz and 50% duty cycle.

```
PWMPort port = new PWMPort(PWMChannel.PWC0, 1000, 50);
```

Methods:

Config: Sets frequency and duty cycle on current PWM channel. Returns true on success

```
public bool Config(
   UInt32 frequency,
   UInt32 dutyCycle
);
```

Start: Starts the PWM channel. Returns true on success, else false

```
public bool Start();
```

Stop: Stops the PWM channel. Returns true on success, else false

```
public bool Stop();
```

4.10 CAN Driver

The AT91SAM9263 Controller Area Network (CAN) controller provides all the features required to implement the serial communication protocol CAN defined by Robert Bosch GmbH, the CAN specification as referred to by
ISO/11898A (2.0 Part A and 2.0 Part B) for high speeds and ISO/11519-2 for low speeds.
The CAN Controller is able to handle all types of frames (Data, Remote, Error and Overload) and achieves a bit rate of 1 Mbit/sec.

CAN controller accesses are made through configuration registers. 16 independent message objects (mailboxes) are implemented. Any mailbox can be programmed as a reception buffer block (even non-consecutive buffers). For the reception of defined messages, one or several message objects can be masked without participating in the buffer feature. An interrupt is generated when the buffer is full. According to the mailbox configuration, the first message received can be locked in the CAN controller registers until the application acknowledges it, or this message can be discarded by new received messages. Any mailbox can be programmed for transmission. Several transmission mailboxes can be enabled in the same time. A priority can be defined for each mailbox independently. The CAN block diagram is shown in figure 4.2

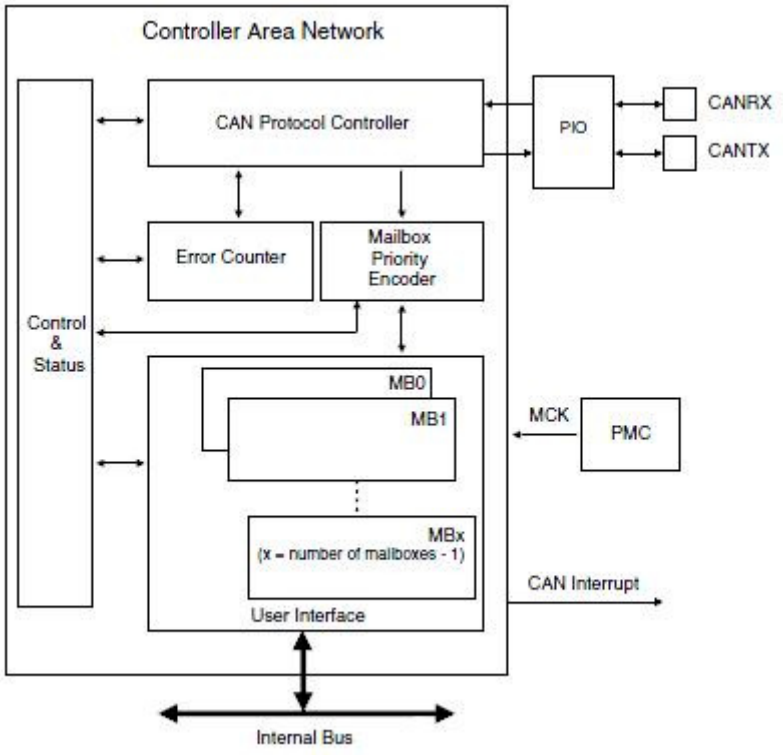


Figure 4.2

4.10.1 Implementation

The driver is implemented as a stream driver, and the driver is accessed through a Win32 file IO interface in WinCE. A managed driver is available that can be used in CF and VS to handle CAN messages.

4.10.2 Managed Driver

The driver is an assembly developed by Proelec that can be used in both C# and VB.NET.
The assembly help file is available in the distribution package, but can also be downloaded from the Proelec homepage.

Filename: PEWinCECANLib.dll
Namespace: Proelec.Win32CE.IO.CAN
Help file: PEWinCECANLib.chm

4.10.2.1 CANMessage

A class that represents a CAN message.

Constructor:

```
[C#]
public void CANMessage();

public void CANMessage(
   byte[] message,
   int position
);

public void CANMessage(
   UInt32 identifier,
   UInt32 length,
   byte[] data
);
```

Parameters:
message : A byte array of CANMessage objects
position: *Can m*essage position in the byte array of CANMessage objects
identifier : The message identifier, 11/29 bits
length: Length of data, 1-8 bytes
data: A byte array of message data, max 8 bytes

Properties:

Identifier: Returns or sets the message identifier

```
public UInt32 Identifier {get; set;}
```

Data: Returns or sets the array of message data, max 8 bytes

```
public byte[] Data {get; set;}
```

Length: Returns or sets the length of message data, 1-8 bytes

```
public UInt32 Length {get; set;}
```

NumOfBytes: Returns the object size in bytes
```
public static int NumOfBytes {get;}
```

Methods:
ToBytes: Returns the message object as an byte array
```
public byte[] ToBytes();
```

4.10.2.2 Config

A class used to configure the CAN controller.

Constructor:
```
[C#]
public void Config();
```

Methods:
SetBaudrate: Sets the baudrate, max 1 Mbits/sec
```
public void SetBaudRate(
   BaudRate baudRate
);
```

baudRate: enum { BITRATE_50, BITRATE_100, BITRATE_125, BITRATE_250, BITRATE_500, BITRATE_800, BITRATE_1000}

SetFrameMode Sets the frame mode
```
public void SetFrameMode(
   FrameMode frameMode
);
```

frameMode: enum { Standard, Extended}

SetWriteInterval: Sets the minimal time between two sent messages, 0 = Disabled
```
public void SetWriteInterval(
   int milliSeconds
);
```

SetPriority256: Sets the CAN interrupt service thread (IST) priority (0-256), 0 is highest
```
public void SetPriority256(
   int priority
);
```

4.10.2.3 TransmitMailbox

A class that represents a mailbox configured for transmission of CAN messages.

Constructor:

```
[C#]
public void TransmitMailbox(
   int mailbox,
);

public void TransmitMailbox(
   int mailbox,
   int identifier
);
```

Parameters:

mailbox : The mailbox index to use, 0-15

identifier: *Can m*essage identifier, 11/29 bits

Properties:

Enabled: Returns or sets the mailbox enabled property

```
public bool Enabled {get; set;}
```

Identifier: Returns or sets the mailbox identifier

```
public int Identifier {get; set;}
```

IsOpen: Returns true if mailbox is opened, else false

```
public bool IsOpen {get;}
```

Methods:

Open: Opens the mailbox

```
public void Open();

public void Open(
   bool enabled
);
```

Close: Closing the mailbox

```
public void Close();
```

Write: Writes CAN message

```
public int Write(
   CANMessage message
);
```

message: CANMessage object to write

```
public int Write(
   byte[] buffer
);
```

buffer: Byte array of CANMessage objects to write

4.10.2.4 SimpleReceptionMailbox

A class that represents a single mailbox configured for reception of CAN messages.
Using this type of mailbox in real-time systems is not recommended. The mailbox can be overloaded with incoming messages. The ChainedReceptionMailbox object should be used in that kind of systems.

Constructor:

```
[C#]
public void SimpleReceptionMailbox(
   int mailbox,
);

public void SimpleReceptionMailbox(
   int mailbox,
   int acceptanceCode,
   int acceptanceMask,
);

public void SimpleReceptionMailbox(
   int mailbox,
   int maxBufferSize
);

public void SimpleReceptionMailbox(
   int mailbox,
   int acceptanceCode,
   int acceptanceMask,
   int maxBufferSize
);
```

Parameters:
mailbox : The mailbox index to use, 0-15
acceptanceCode : The mailbox acceptance code
acceptanceMask : The mailbox acceptance mask
maxBufferSize: The maximal number of CANMessage objects that can be stored in the buffer

Properties:
Enabled: Returns or sets the mailbox enabled property

```
public bool Enabled {get; set;}
```

AcceptanceCode: Returns or sets the mailbox acceptance code

```
public int AcceptanceCode {get; set;}
```

AcceptanceMask: Returns or sets the mailbox acceptance mask

```
public int AcceptanceMask {get; set;}
```

Count: Returns the number of messages in mailbox buffer

```
public int Count {get;}
```

IsOpen: Returns true if mailbox is opened, else false
```
public bool IsOpen {get;}
```

Methods:

Open: Opens the mailbox
```
public void Open();

public void Open(
   bool enabled
);
```

Close: Closing the mailbox
```
public void Close();
```

Clear: Clears all the messages in the mailbox buffer
```
public void Clear();
```

Read: Reads a CAN message
```
public CANMessage Read();

public int Read(
   byte[] buffer,
   int count
);
```

buffer: Byte array of CAN messages to write
count: Number of messages in buffer

Events:

OnMessageRecieved: Event is trigged when a new message arrives in the buffer
```
public event MessageRecievedEventHandler OnMessageRecieved;
```

MessageRecievdArgs: The event argument constructor
```
public void MessageRecievedArgs(
   CANMessage message
);
```

Message property: Returns the CANMessage object
```
public CANMessage Message {get;}
```

4.10.2.5 ChainedReceptionMailbox

A class that represents two or more chained mailboxes configured for reception of CAN messages. This type of mailbox is recommended to use in real-time systems.

Constructor:

```
[C#]
public void ChainedReceptionMailbox(
   int startMailbox,
   int numOfMailboxes
);

public void ChainedReceptionMailbox(
   int mailbox,
   int numOfMailboxes,
   int acceptanceCode,
   int acceptanceMask,
);

public void ChainedReceptionMailbox(
   int mailbox,
   int numOfMailboxes,
   int maxBufferSize
);

public void ChainedReceptionMailbox(
   int mailbox,
   int numOfMailboxes,
   int acceptanceCode,
   int acceptanceMask,
   int maxBufferSize
);
```

Parameters:

mailbox : The start mailbox index to use, 0-15

numOfMailboxes : The number of mailboxes to chain

acceptanceCode : The mailbox acceptance code

acceptanceMask : The mailbox acceptance mask

maxBufferSize: The maximal number of CANMessage objects that can be stored in the buffer

Properties:

Enabled: Returns or sets the mailbox enabled property

```
public bool Enabled {get; set;}
```

AcceptanceCode: Returns or sets the mailbox acceptance code

```
public int AcceptanceCode {get; set;}
```

AcceptanceMask: Returns or sets the mailbox acceptance mask

```
public int AcceptanceMask {get; set;}
```

Count: Returns the number of messages in mailbox buffer

```
public int Count {get;}
```

IsOpen: Returns true if mailbox is opened, else false
```
public bool IsOpen {get;}
```

Methods:

Open: Opens the mailbox
```
public void Open();

public void Open(
   bool enabled
);
```

Close: Closing the mailbox
```
public void Close();
```

Clear: Clears all the messages in the mailbox buffer
```
public void Clear();
```

Read: Reads a CAN message
```
public CANMessage Read();

public int Read(
   byte[] buffer,
   int count
);
```

buffer: Byte array of CAN messages to write
count: Number of messages in buffer

Events:

OnMessageRecieved: Event is trigged when a new message arrives in the buffer
```
public event MessageRecievedEventHandler OnMessageRecieved;
```

MessageRecievdArgs: The event argument constructor
```
public void MessageRecievedArgs(
   CANMessage message
);
```

Message property: Returns the CANMessage object
```
public CANMessage Message {get;}
```

4.11 TWI/I2C Driver

The Two-wire Interface (TWI) interconnects components on a unique two-wire bus, made up of one clock line and one data line with speeds of up to 400 Kbits per second, based on a byte-oriented transfer format. It can be used with any Atmel Two-wire Interface bus Serial EEPROM and Inter-Integrated Circuit (I2C) compatible device such as Real Time Clock (RTC), Dot Matrix/Graphic LCD Controllers and Temperature Sensor, to name but a few. The TWI is programmable as master transmitter or master receiver with sequential or single-byte access. A configurable baud rate generator permits the output data rate to be adapted to a wide range of core clock frequencies. Table 4.1 lists the compatibility level of the Atmel Two-wire Interface and a full I2C compatible device. Figure 4.3 shows the TWI block diagram.

Table 4.1

I2C Standard	Atmel TWI
Standard Mode Speed (100 KHz)	Supported
Fast Mode Speed (400 KHz)	Supported
7 or 10 bits Slave Addressing	Supported
START BYTE, START + b000000001 + Ack + Sr	Not Supported
Repeated Start (Sr) Condition	Not Fully Supported
ACK and NACK Management	Supported
Slope control and input filtering (Fast mode)	Supported
Clock strectching	Supported

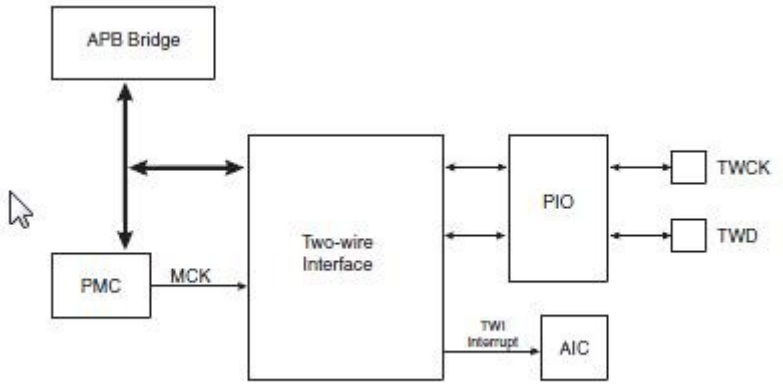


Figure 4.3

4.11.1 Implementation

The driver is implemented as a stream driver, and the driver is accessed through a Win32 file IO interface in WinCE. A managed driver is available that can be used in CF and VS to handle the I2C bus.

4.11.2 Managed Driver

The driver is an assembly developed by Proelec that can be used in both C# and VB.NET.
The assembly help file is available in the distribution package, but can also be downloaded from the Proelec homepage.

Filename: PEWinCEI2CLib.dll
Namespace: Proelec.Win32CE.IO.I2C
Help file: PEWinCEI2CLib.chm

4.11.2.1 I2CBus

A class used to handle the TWI/I2C bus.

***Constructor*:**

```
[C#]
public void I2CBus();

public void I2CBus(
   UInt32 clockRate
);
```

Parameters:
clockRate : The bus clock rate. Default = 100K.

Methods:
SetTransferRate: Sets the clock rate

```
public void SetTransferRate(UInt32 clockRate);
```

Read: Reads data from a device connected to the bus, exceptions are thrown on errors

```
public void Read(
   UInt32 deviceAddr,
   UInt32 internalAddr,
   InternalAddressSize internalAddrSize,
   bool disableInterrupt,
   byte[] buffer,
   UInt32 bytesToRead
);
```

Parameters:
deviceAddr : Device address
internalAddr : Internal address
internalAddrSize : Internal address size, enum: {NoAddress, OneByte, TwoByte, ThreeByte}
disableInterrupt : Interrupt Enable/Disable while reading
buffer: Data buffer
bytesToRead : Number of bytes to read

`Write: Write data to a device connected to the bus`, exceptions are thrown on errors

```
public void Write(
   UInt32 deviceAddr,
   UInt32 internalAddr,
   InternalAddressSize internalAddrSize,
   bool disableInterrupt,
   byte[] buffer,
   UInt32 bytesToWrite
);
```

Parameters:

deviceAddr :	Device address
internalAddr :	Internal address
internalAddrSize :	Internal address size, enum: {NoAddress, OneByte, TwoByte, ThreeByte}
disableInterrupt :	Interrupt Enable/Disable while writing
buffer:	Data buffer
bytesToWrite :	Number of bytes to write

4.12 SPI Driver

The Serial Peripheral Interface (SPI) circuit is a synchronous serial data link that provides communication with external devices in Master or Slave Mode. It also enables communication between processors if an external processor is connected to the system.
The Serial Peripheral Interface is essentially a shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master”' which controls the data flow, while the other devices act as “slaves'' which have data shifted into and out by the master. Different CPUs can take turn being masters (Multiple Master Protocol opposite to Single Master Protocol where one CPU is always the master while all of the others are always slaves) and one master may simultaneously shift data into multiple slaves. However, only one slave may drive its output to write data back to the master at any given time.
A slave device is selected when the master asserts its NSS signal. If multiple slave devices exist, the master generates a separate slave select signal for each slave (NPCS).

The SPI system consists of two data lines and two control lines:

- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input(s) of the slave(s).
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master. There may be no more than one slave transmitting data during any particular transfer.
- Serial Clock (SPCK): This control line is driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of baud rates; the SPCK line cycles once for each bit that is transmitted.
- Slave Select (NSS): This control line allows slaves to be turned on and off by hardware

Block Diagram:

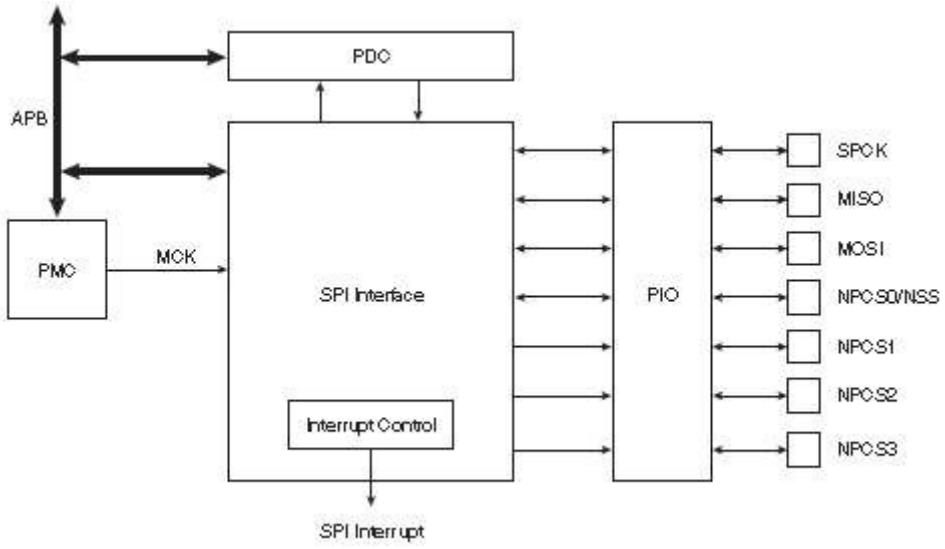


Figure 4.4

Application Block Diagram: Single Master/Multiple Slave Implementation

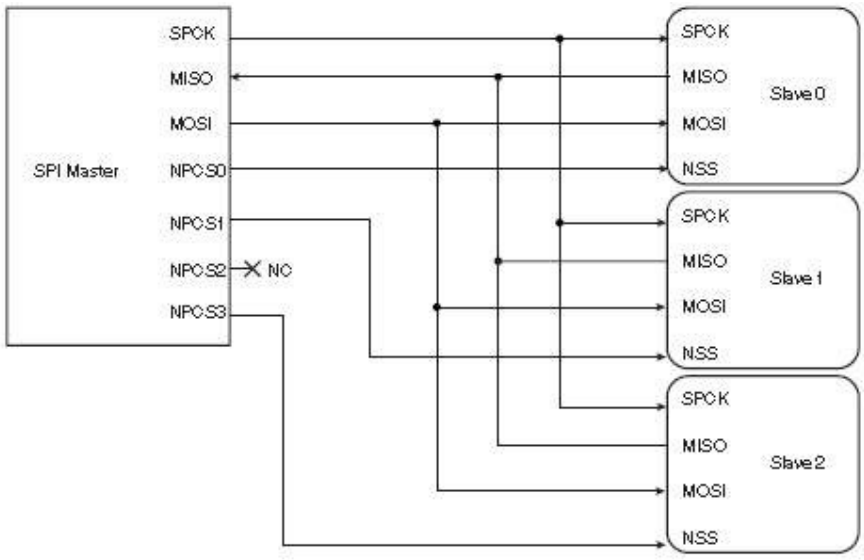


Figure 4.5

4.12.1 Implementation

The driver is implemented as a stream driver, and the driver is accessed through a Win32 file IO interface in WinCE.

4.12.2 Managed Driver

A public managed SPI driver is not available. Please contact Proelec to get more information about how to get a custom driver that you can use in VS.

5 Linux

5.1 General Information

The GEMSTONE platform can be used with an embedded Linux distribution.
Proelec can develop a custom Linux image. The image can be customized to support all type of functions and peripherals on the board. Please contact Proelec for more information about building a customized Linux image.

6 FlashTool

6.1 General Information

FlashTool is a program used to program both the DataFlash and NandFlash memory on the board. The tool exists only for use with the WinCE distribution from Proelec, but can be customized for other distributions or environments.
The program uses the SAM Boot Assistant (SAM-BA) Application Program Interface (API) from Atmel to interact with the board memories.

6.2 Windows CE

The program FlashTool AT91SAM9263CE is distributed with the Windows CE image, and is used to put the FirstBoot, EBoot and OS-image into the board memories.

DataFlash: FirstBoot and EBoot
NandFlash: OS-image

6.2.1 Program GUI

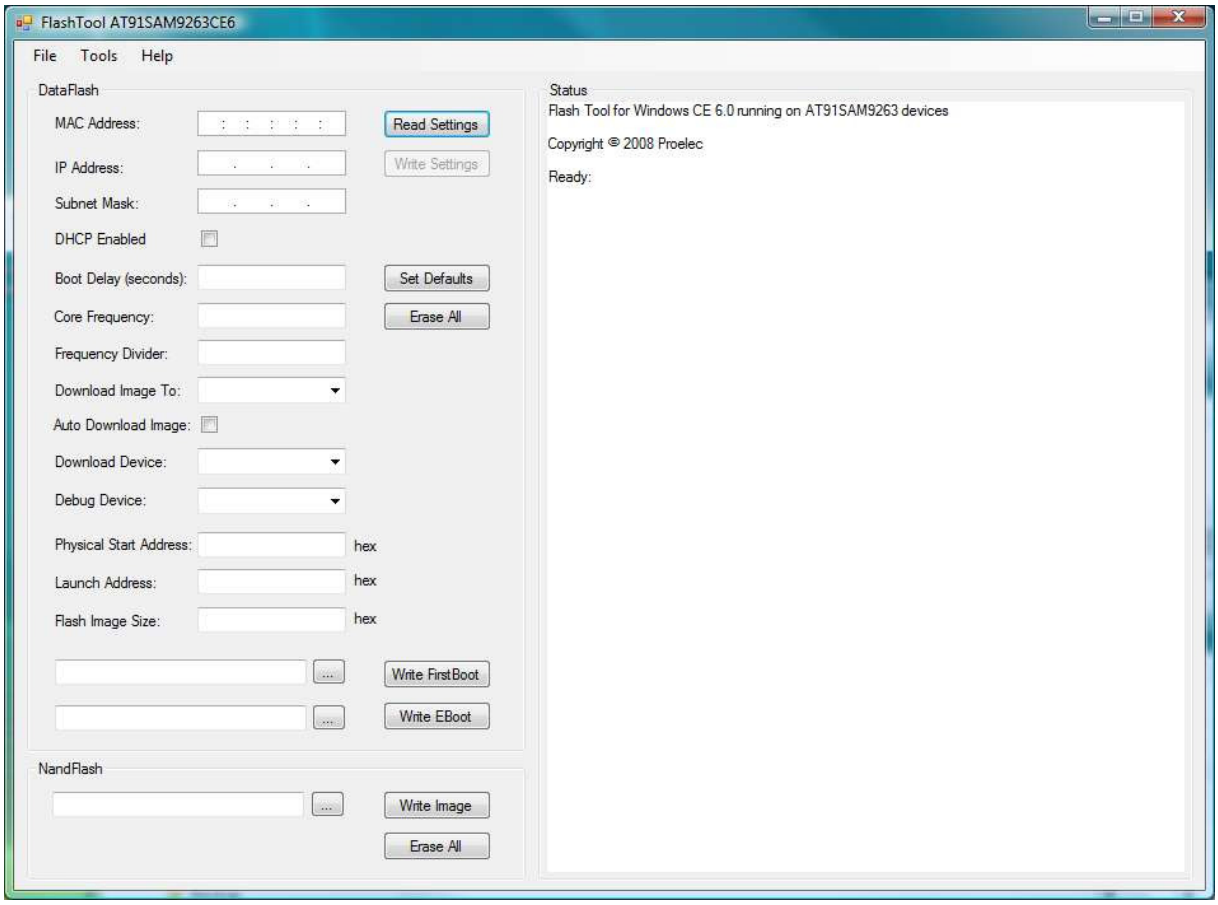


Figure 6.1

6.2.2 Install SAM-BA Windows Driver

If the distribution package AT91-ISP 1.1X from Atmel already is installed on your computer, you don't need to install the custom FlashTool driver from Proelec if the version of the package is 1.10 or higher. If you have an older version you must uninstall the AT91-ISP 1.1X package before installing the driver from Proelec.

If you want to install the FlashTool driver from Proelec, just go to the *Tools* menu in the program and select *install driver,* See figure 6.2. The driver is not Microsoft certified, so you must accept the installation when the acceptance dialog appears on the screen.
When installation of the driver is finished the board can be started up in SAM-BA (programming) mode, see 6.2.3.

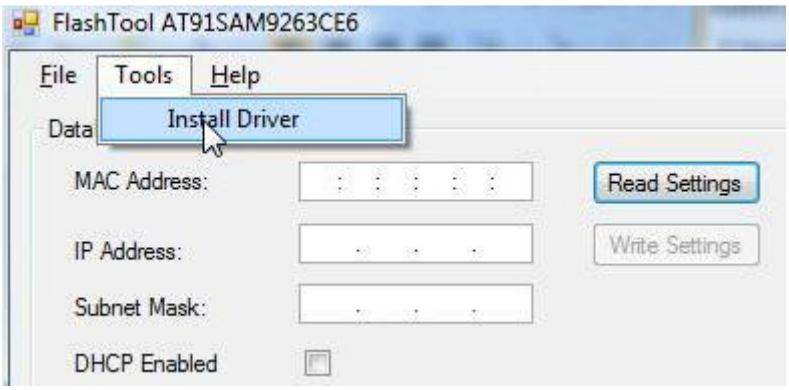


Figure 6.2

6.2.3 Starting up the board in SAM-BA Mode

Hold down button U6 (see figure 2.1) while switching on the board power supply (5V).
If you hear a sound from the PC, the board should be in SAM-BA (programming) mode.
If you have problems to set the board in programming mode, check that the device exists in the Windows device manager. If you have installed the Proelec driver you should look in the Ports (COM and LPT) category of devices, and if you have installed the AT91-ISP 1.1X package you should look in USB devices.

6.2.4 Programming the DataFlash

The DataFlash memory is programmed in two phases. First phase is to put the FirstBoot file into the memory and second phase the EBoot file. Figure 6.3 shows the controls used while programming the DataFlash.

Figure 6.3

FirstBoot:
Select the FirstBoot file and then use the *Write FirstBoot* button to put the file into memory.

EBoot:
Select the EBoot file and then use the *Write EBoot* button to put the file into memory.

6.2.5 Erasing the DataFlash

The DataFlash memory can be erased if you press the *Erase All* button in the DataFlas section of the program, see figure 6.4. To re-enable booting the OS, you must program the FirstBoot and EBoot file. The EBoot settings are also erased, so you must load *Default Settings*, edit and save the settings to successfully boot the OS again.

Figure 6.4

6.2.6 Programming the NandFlash

The NandFlash memory is programmed by putting the OS-image file into the memory.
Figure 6.5 shows the controls to use while programming. Select the OS-image file and then use the *Write Image* button to put the file into memory. The EBoot field *Flash Image Size* is automatically changed, so remember to save the new setting to DataFlash if the image size is different.

Figure 6.5

6.2.7 Erasing the NandFlash

The NandFlash memory can be erased if you press the *Erase All* button, see figure 6.5.
To re-enable booting the OS, you must re-program the OS-image file into the NandFlash memory.

6.2.8 EBoot Settings

To load the EBoot default settings into the program fields, use the *Set Defaults* button. Figure 6.6 shows the controls to use when handling the settings.
Use the *Read Settings* button to load data from the DataFlash memory and *Write Settings* button to store the data back into the memory.

Figure 6.6

6.2.9 Programming Status

Current programming status and errors are shown in the list box to the right, see figure 6.7.

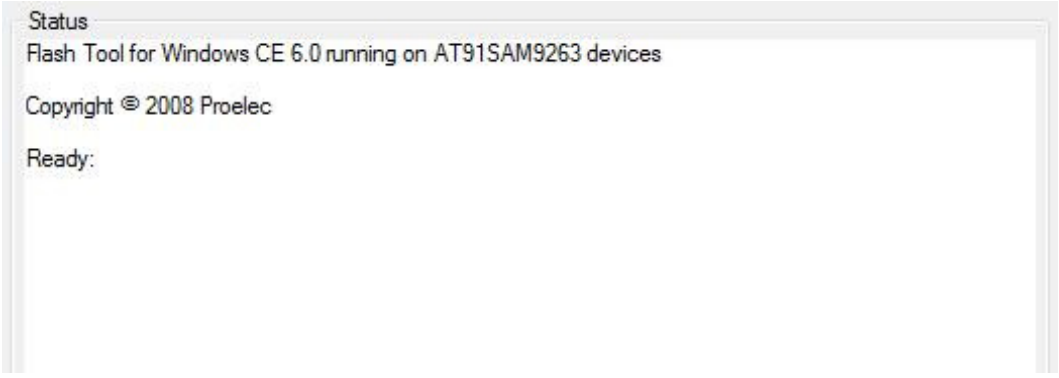
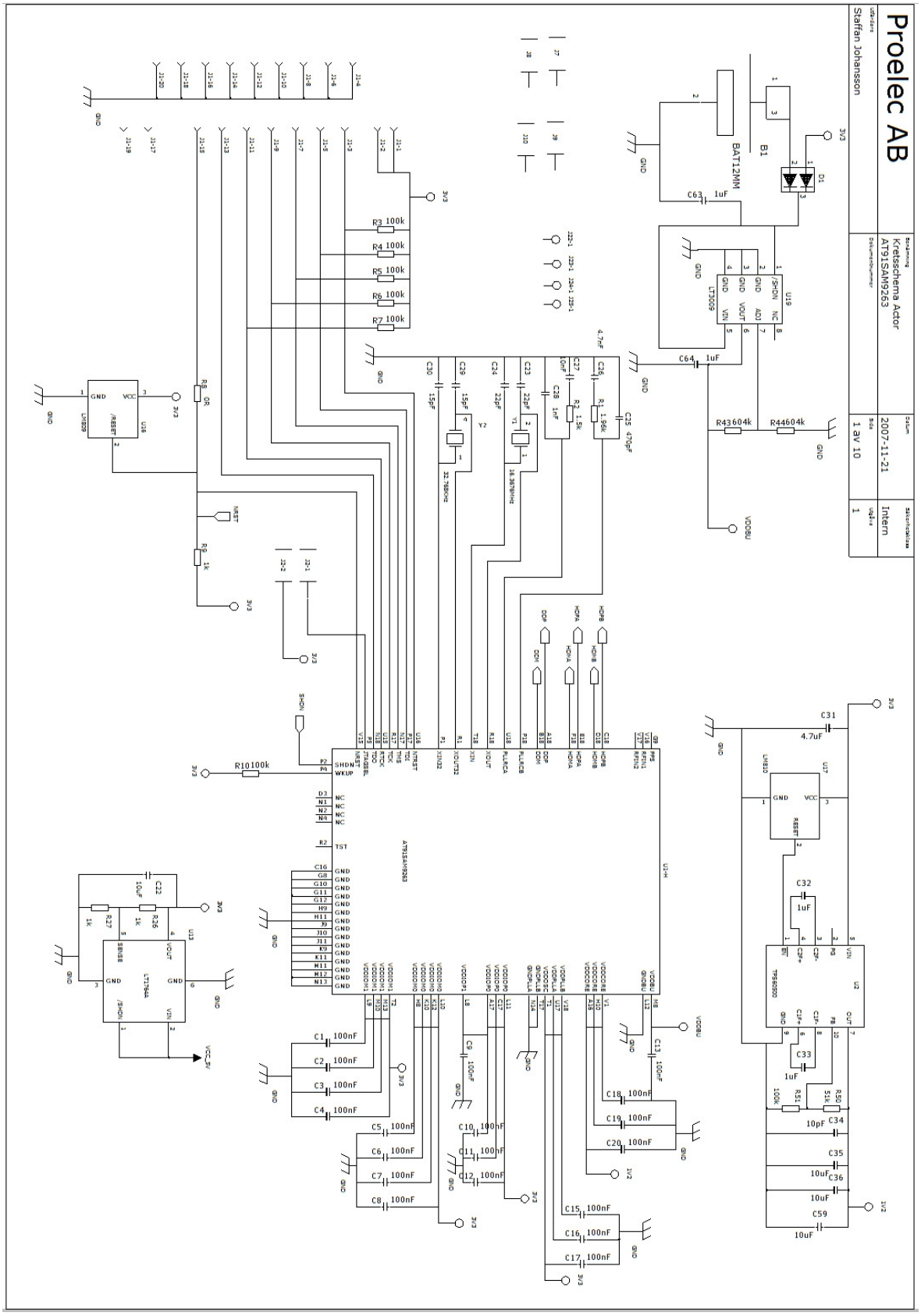


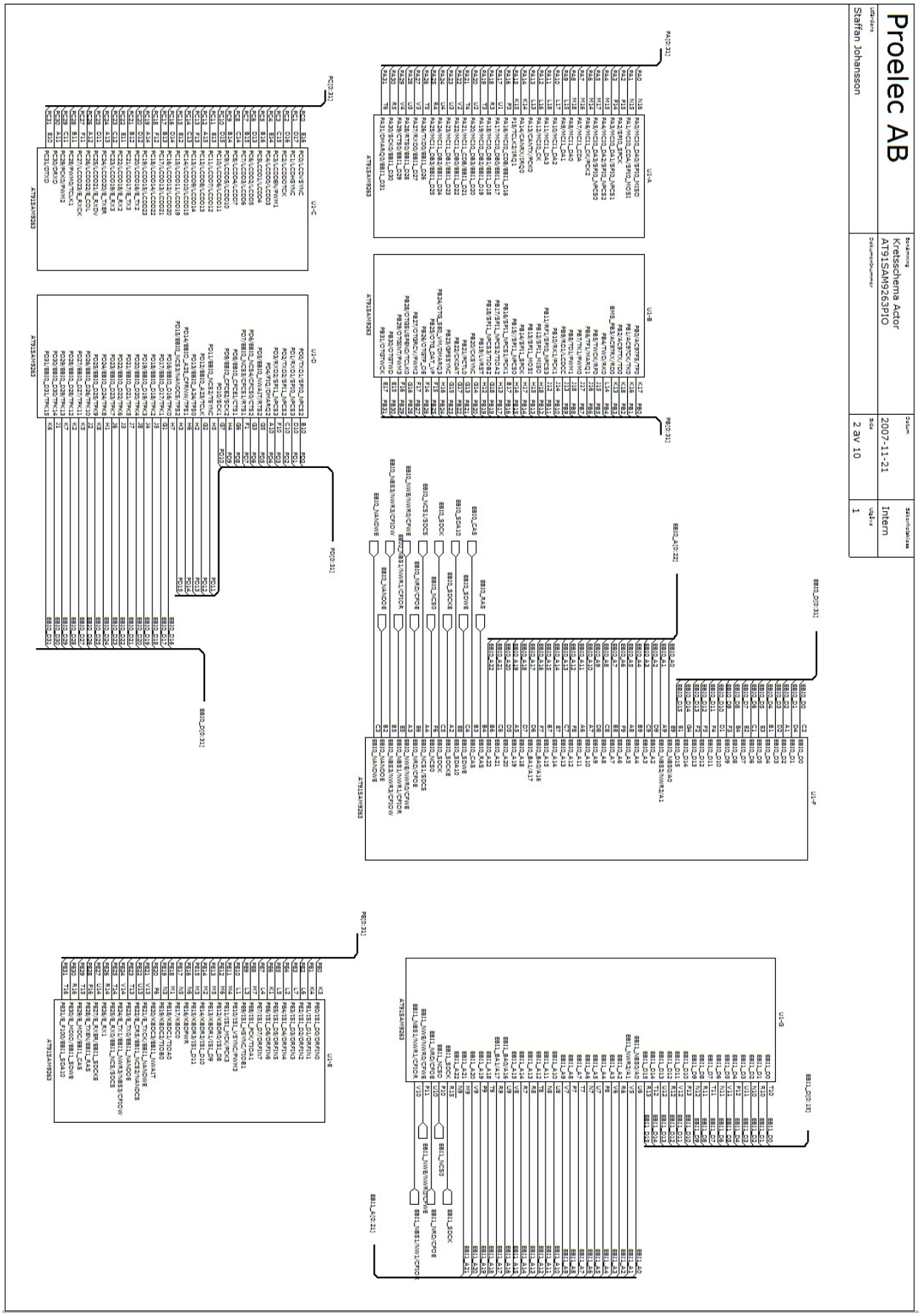
Figure 6.7

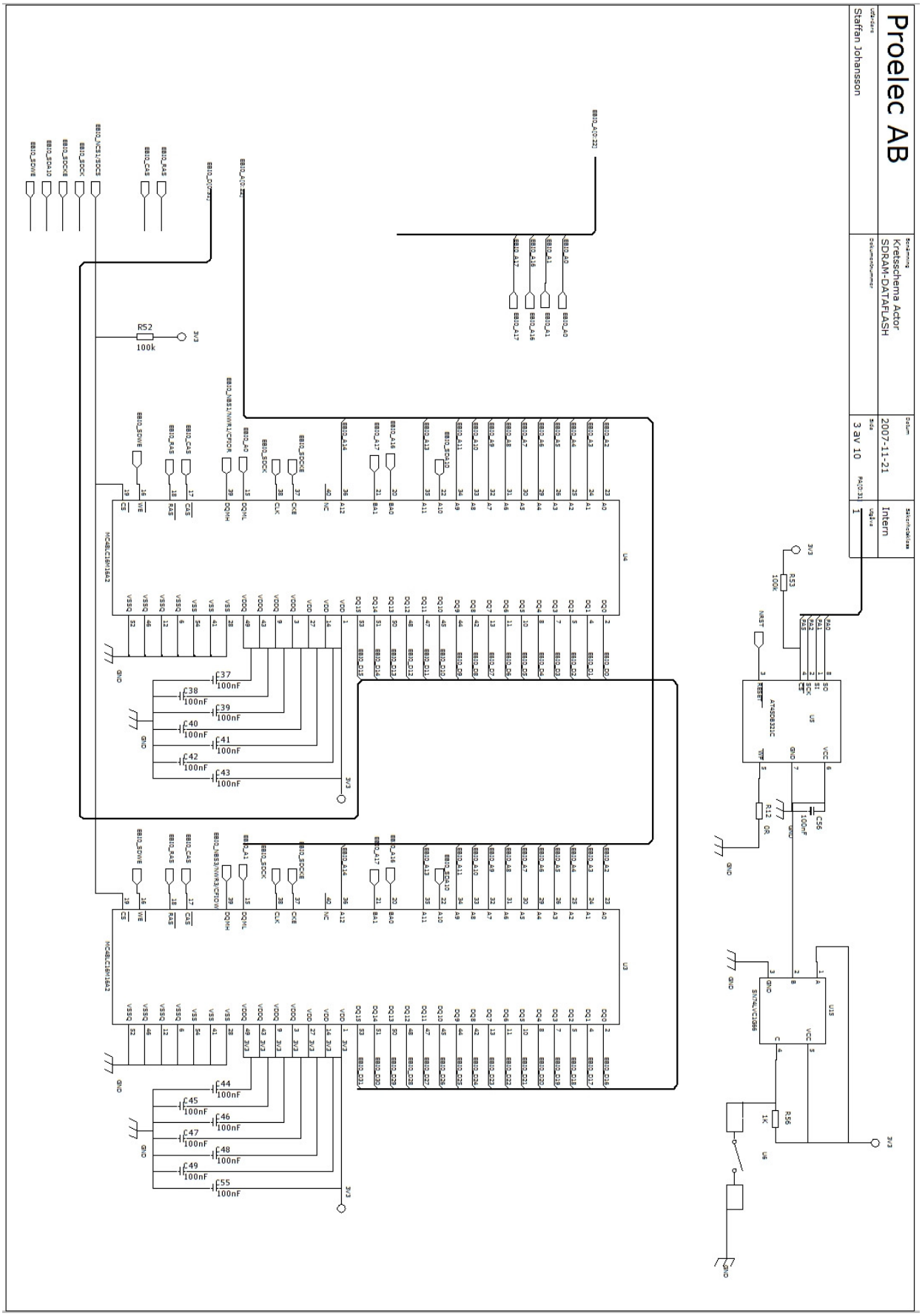
6.3 Linux

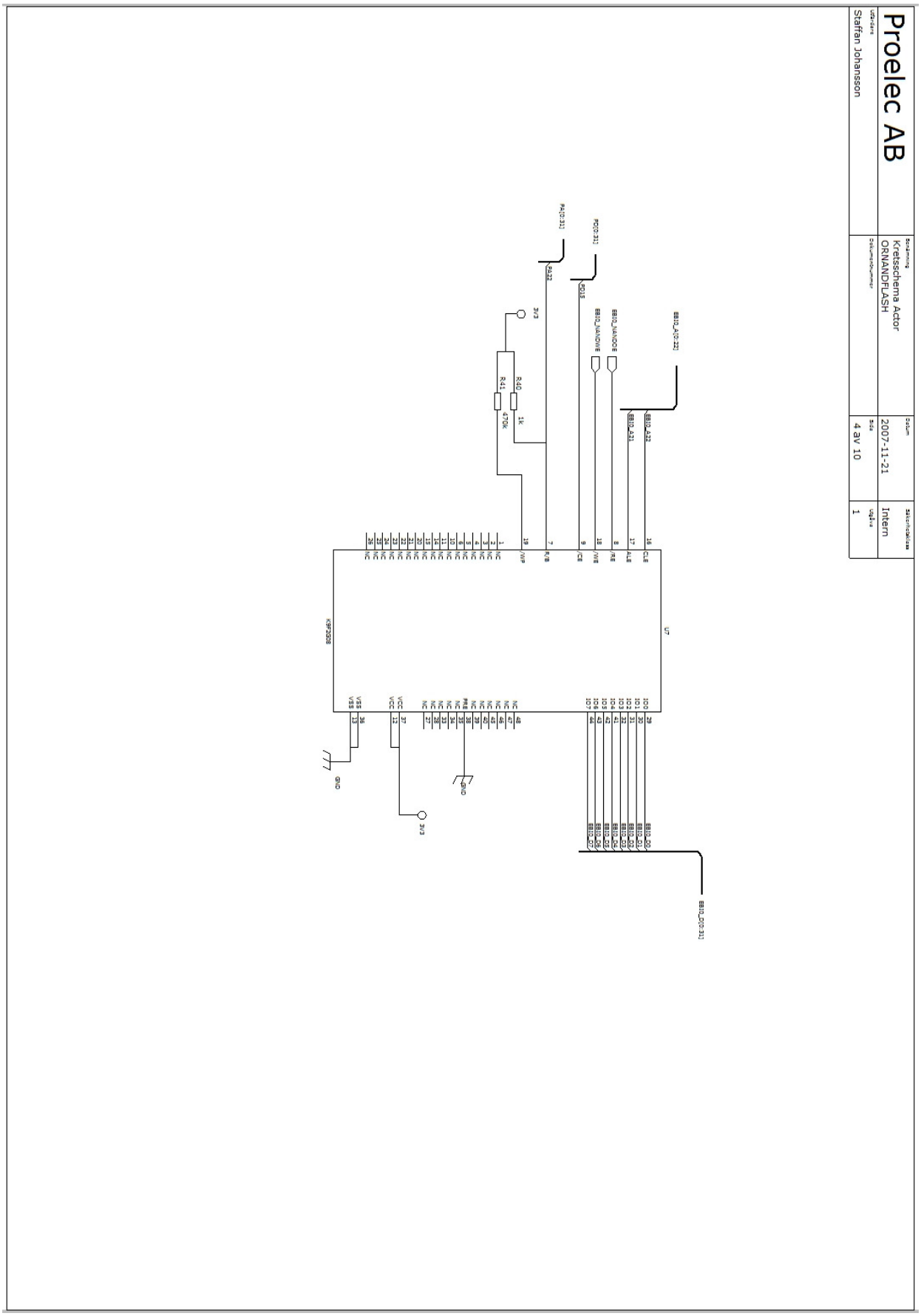
A FlashTool program for a custom embedded Linux distribution can be made by Proelec.
Please contact Proelec for more information.

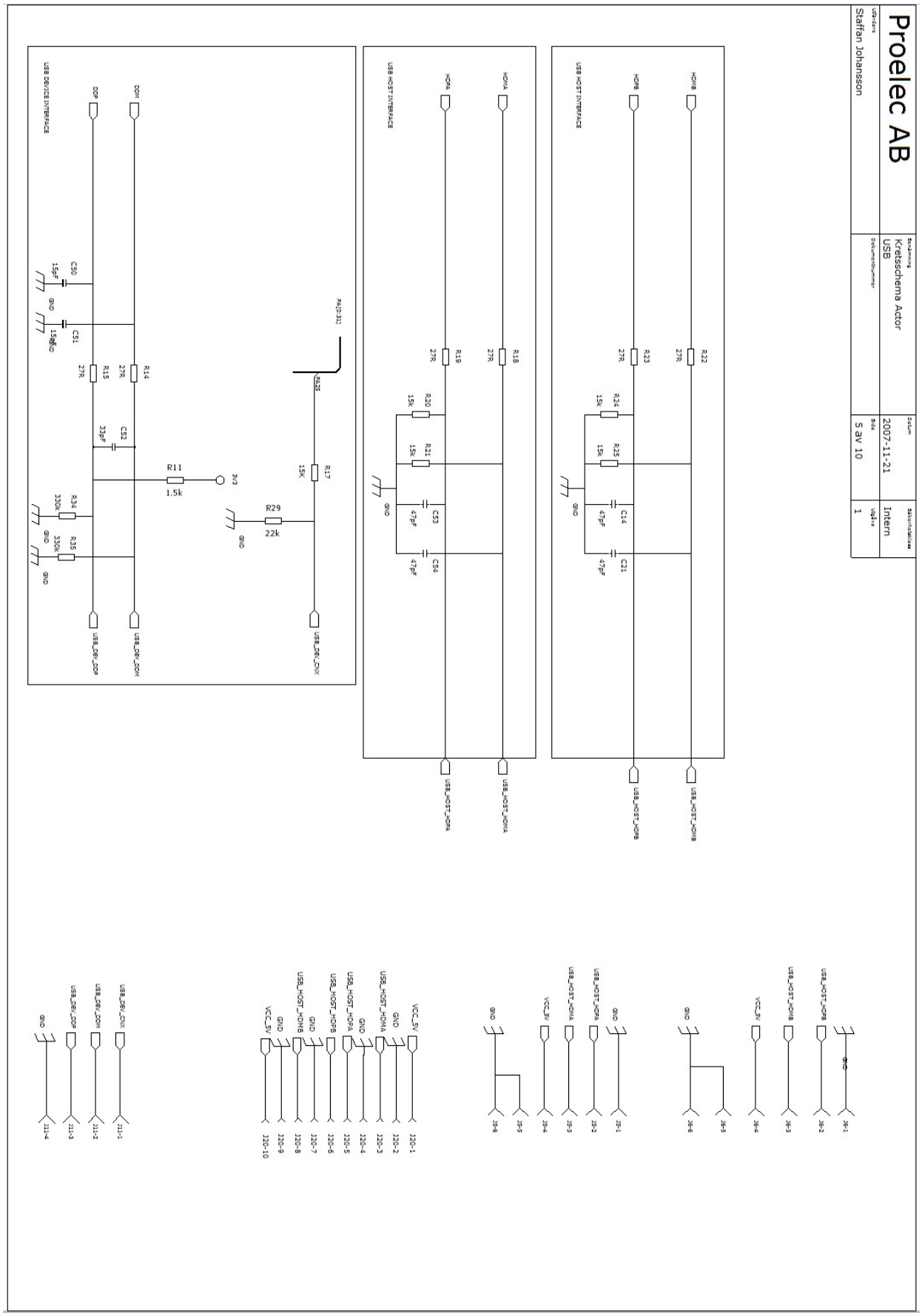
7 Appendix A: Schematics

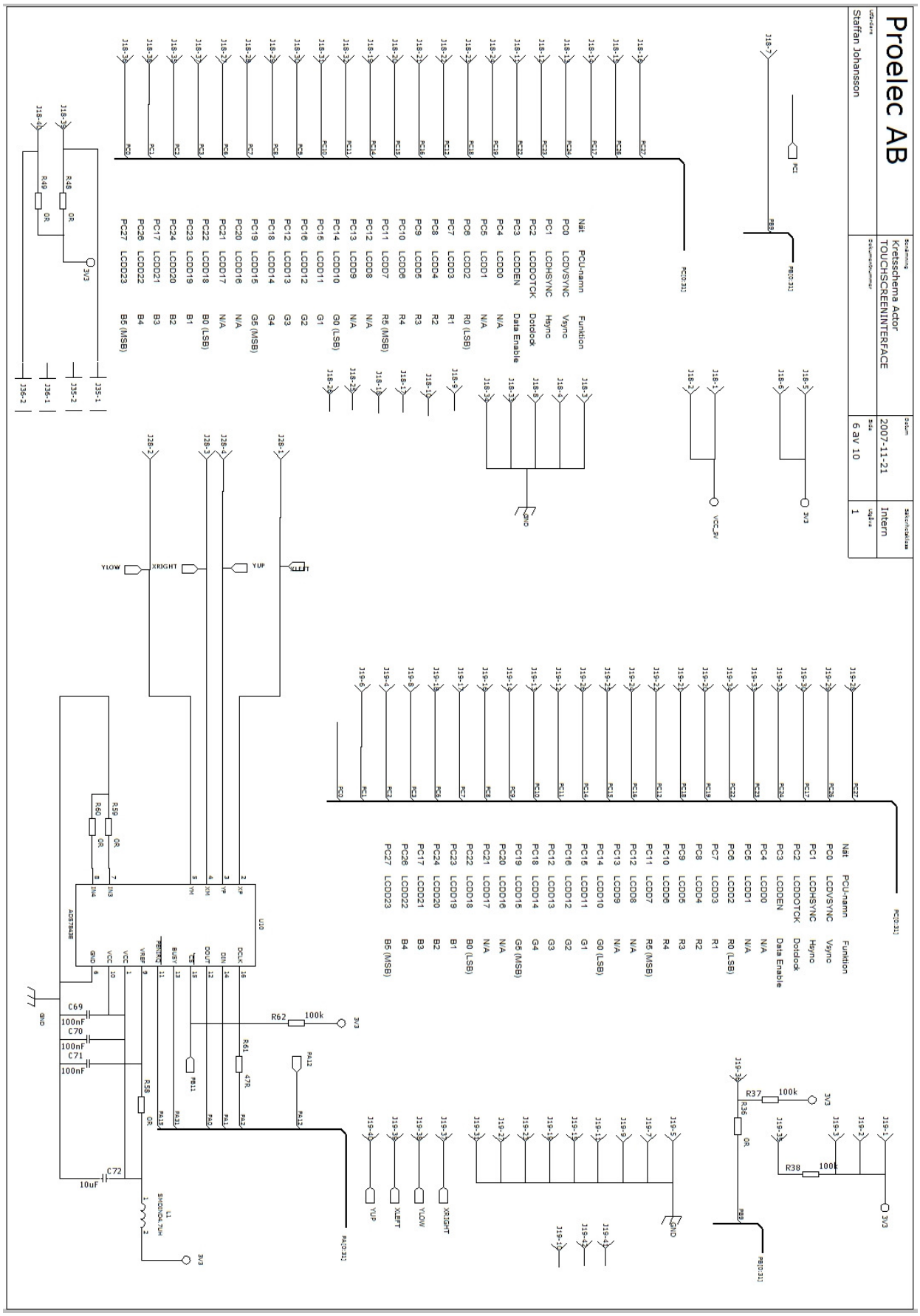


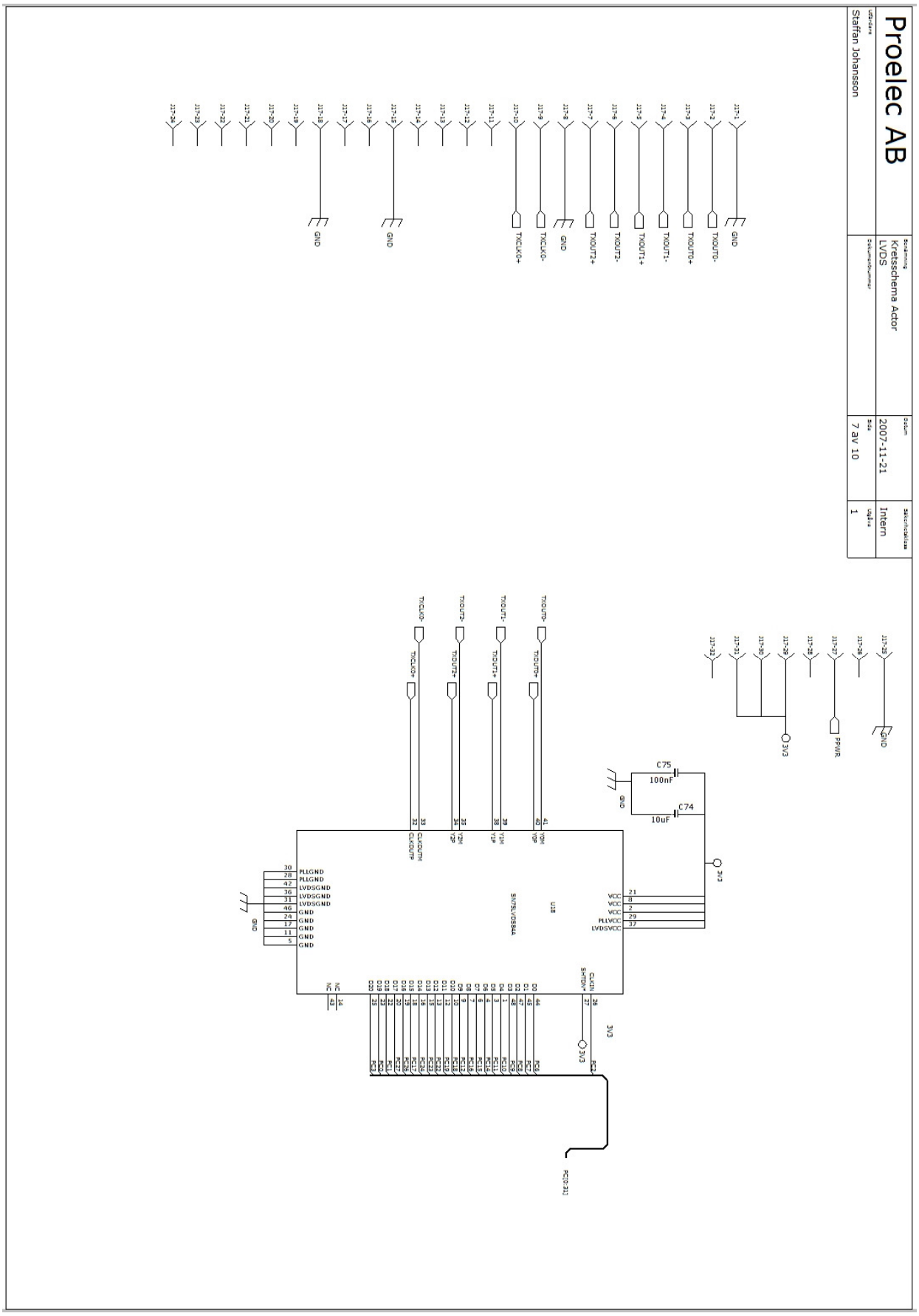

Proelec AB
Kretsschema Actor AT91SAM9263PIO
Staffan Johansson
2007-11-21
Intern
2 av 10
1

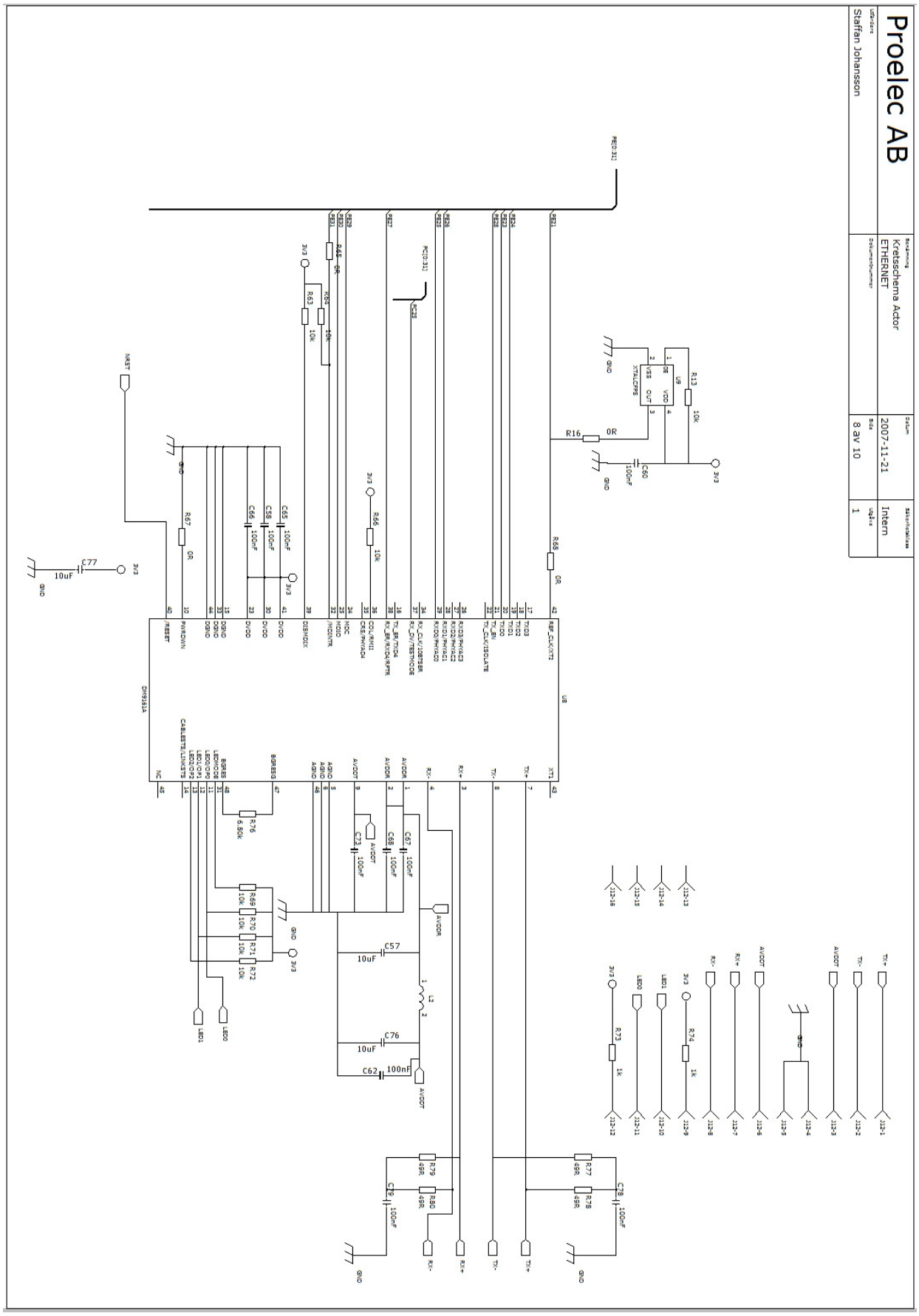

Proelec AB
Kretsschema Actor
SDRAM-DATAFLASH
Staffan Johansson
2007-11-21
3 av 10
Intern
R52
100k
R53
100k
R12
0R
C56
100nF
R56
1K
C37
100nF
C38
100nF
C39
100nF
C40
100nF
C41
100nF
C42
100nF
C43
100nF
C44
100nF
C45
100nF
C46
100nF
C47
100nF
C48
100nF
C49
100nF
C55
100nF
3V3
GND

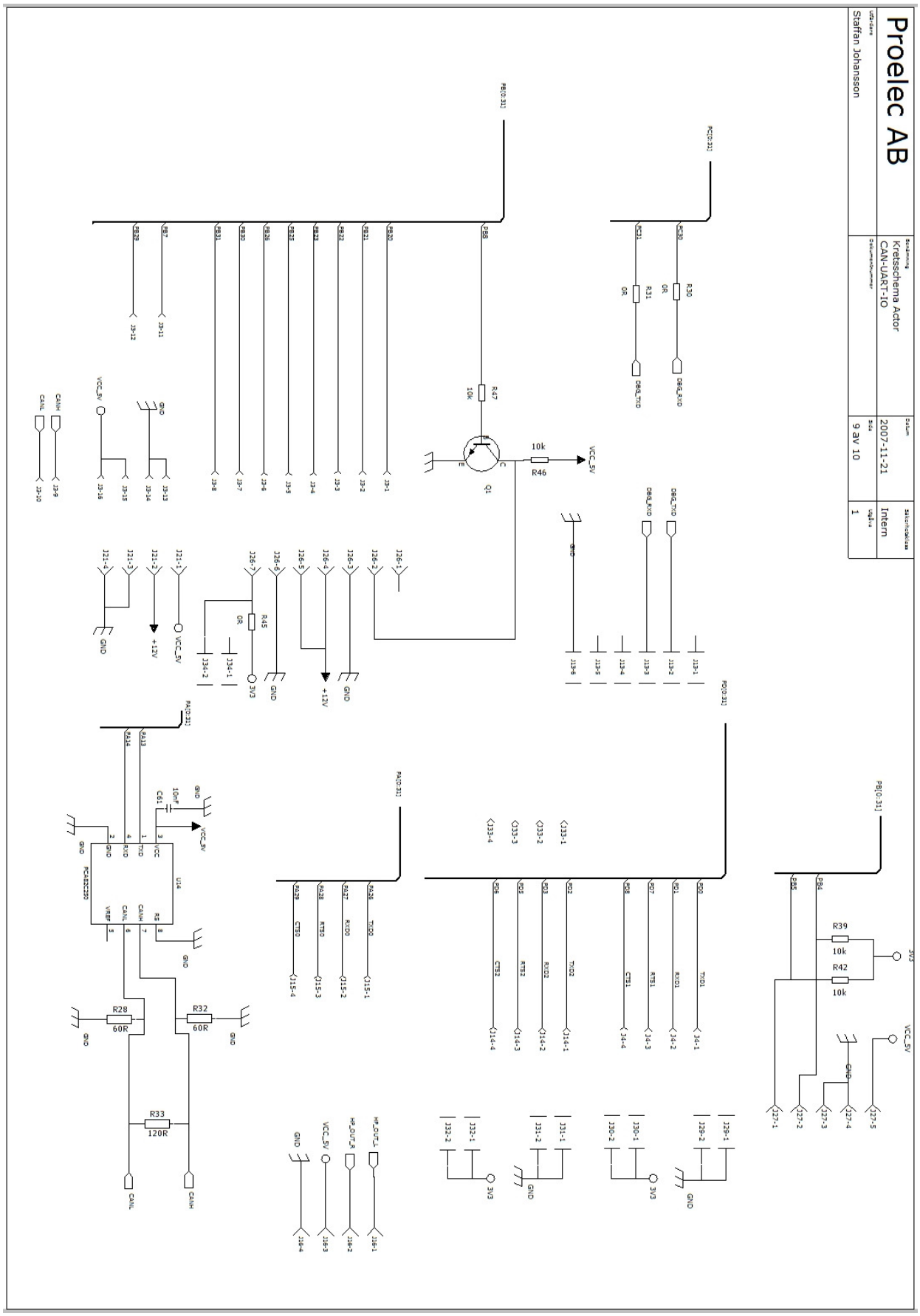

Proelec AB
Kretsschema Actor
ORNANDFLASH
Staffan Johansson
2007-11-21
Intern
4 av 10
1
R40
R41
1k
470k
3V3
GND
U7
K9F2808

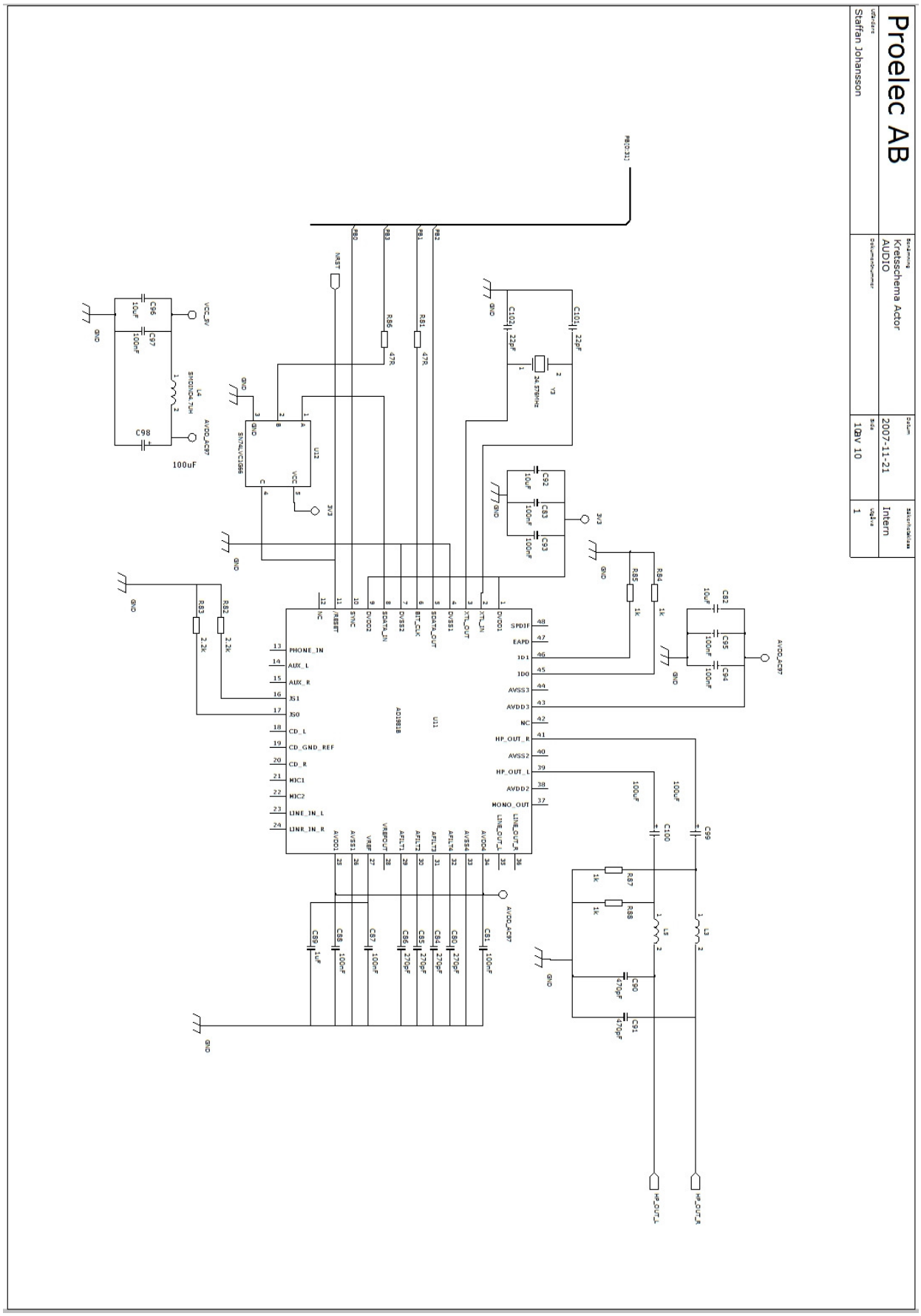

Proelec AB
Staffan Johansson
Kretsschema Actor USB
2007-11-21
5 av 10
Intern
1
USB_HOST_HDMB
USB_HOST_HDPB
USB_HOST_HDMA
USB_HOST_HDPA
USB_DEV_CLK
USB_DEV_DDM
USB_DEV_DDP
VCC_5V
GND
R22
R23
R24
R25
C14
C21
R18
R19
R20
R21
C53
C54
R17
R29
R11
R14
R15
C50
C51
C52
R34
R35
PA20,31
3V3
15k
27R
47pF
33pF
22k
1.5k
330k
15pF
J6-1
J6-2
J6-3
J6-4
J6-5
J6-6
J5-1
J5-2
J5-3
J5-4
J5-5
J5-6
J20-1
J20-2
J20-3
J20-4
J20-5
J20-6
J20-7
J20-8
J20-9
J20-10
J11-1
J11-2
J11-3
J11-4


Proelec AB
Kretsschema Actor
TOUCHSCREENINTERFACE
Staffan Johansson
2007-11-21
6 av 10
Intern
1
Nät PCU-namn Funktion
PC0 LCDVSYNC Vsync
PC1 LCDHSYNC Hsync
PC2 LCDDOTCK Dotclock
PC3 LCDDEN Data Enable
PC4 LCDD0 N/A
PC5 LCDD1 N/A
PC6 LCDD2 R0 (LSB)
PC7 LCDD3 R1
PC8 LCDD4 R2
PC9 LCDD5 R3
PC10 LCDD6 R4
PC11 LCDD7 R5 (MSB)
PC12 LCDD8 N/A
PC13 LCDD9 N/A
PC14 LCDD10 G0 (LSB)
PC15 LCDD11 G1
PC16 LCDD12 G2
PC12 LCDD13 G3
PC18 LCDD14 G4
PC19 LCDD15 G5 (MSB)
PC20 LCDD16 N/A
PC21 LCDD17 N/A
PC22 LCDD18 B0 (LSB)
PC23 LCDD19 B1
PC24 LCDD20 B2
PC17 LCDD21 B3
PC26 LCDD22 B4
PC27 LCDD23 B5 (MSB)
R48 0R
R49 0R
R50 0R
R55 0R
R58 0R
R61 47R
R62 100k
R36 0R
R37 100k
R38 100k
C69 100nF
C70 100nF
C71 100nF
C72 10uF
U10
ADS7846E
L1
BMH0805A-7UH
YLOW
XRIGHT
YUP
XLEFT
3V3
GND
VCC_5V


Proelec AB
Staffan Johansson
Kretsschema Actor LVDS
2007-11-21
7 av 10
Intern
1
GND
TXOUT0-
TXOUT0+
TXOUT1-
TXOUT1+
TXOUT2-
TXOUT2+
GND
TXCLKO-
TXCLKO+
3V3
PPWR
C75
100nF
C74
10uF
U18
SN75LVDS84A
PLLGND
LVDSGND
VCC
PLLVCC
LVDSVCC
CLKOUTM
CLKOUTP
CLKIN
SHTDN*


Proelec AB
Kretsschema Actor
ETHERNET
Staffan Johansson
2007-11-21
8 av 10
1
Intern
DM9161A
XTALOPPS
R16
0R
C60
100nF
R13
10k
C77
10uF
R67
0R
C65
100nF
C58
100nF
C66
100nF
R63
10k
R65
0R
R66
10k
R68
0R
C76
10uF
C62
100nF
C57
10uF
C73
100nF
C67
100nF
R76
6.80k
R69
10k
R70
10k
R71
10k
R72
10k
R73
1k
R74
1k
R79
49R
R80
49R
R77
49R
R78
49R
C79
100nF
C78
100nF
LED1
LED0
AVDDR
AVDDT
3V3
GND
RX+
RX-
TX+
TX-


Proelec AB
Kretsschema Actor
CAN-UART-IO
Staffan Johansson
2007-11-21
9 av 10
Intern
1
R47
10k
Q1
R46
10k
VCC_5V
R45
0R
3V3
GND
+12V
R39
10k
R42
10k
R28
60R
R32
60R
R33
120R
CANH
CANL
DBG_RXD
DBG_TXD
HP_OUT_L
HP_OUT_R


Proelec AB
Staffan Johansson
Kretsschema Actor AUDIO
2007-11-21
Rev 10
Intern
1
ADI1881B
U11
PHONE_IN
AUX_L
AUX_R
JS1
JS0
CD_L
CD_GND_REF
CD_R
MIC1
MIC2
LINE_IN_L
LINR_IN_R
HP_OUT_R
HP_OUT_L
MONO_OUT
LINEOUT_L
LINEOUT_R
SPDIF
EAPD
GPIO1
GPIO0
AVDD3
AVDD2
AVSS3
AVSS2
AVSS1
AVDD1
AVSS4
AVDD4
VREFOUT
VREF
AFILT1
AFILT2
AFILT3
AFILT4
DVDD1
DVSS1
DVDD2
DVSS2
SDATA_OUT
SDATA_IN
BIT_CLK
SYNC
RESET
NC
XTL_OUT
XTL_IN
3V3
GND
AVDD_AC97
VCC_5V
100uF
22pF
24.576MHz
470R
2.2k
1k
10uF
100nF
270pF
470pF
1uF

8 Revision History

Version	Comments	Date
1.0	Document start	2009-04-17

9 References

Atmel. (2009). AT91SAM9263 Resources page.
http://www.atmel.com/dyn/products/product_card.asp?part_id=4056. (2009-04-17).

Atmel. (2009). AT91SAM9263 Preliminary.
http://www.atmel.com/dyn/resources/prod_documents/doc6249.pdf. (2009-04-17).